CUBIT 10.0 User Documentation

Table Of Contents

Introduction 1

Key Features 1
- Geometry Creation, Modification, and Healing 1
- Non-Manifold Topology 1
- Geometry Decomposition 2
- Mesh Generation 2
- Boundary Conditions 2
- Element Types 2
- Graphics Display Capabilities 2
- Graphical User Interface 2
- Command Line Interface 2

How to Use This Manual 3

Licensing, Distribution and Installation 3

Hardware Requirements 3

Trademark Notice 4

CUBIT Mailing Lists 4

Environment Control 6

Session Control 6

Starting and Exiting a CUBIT Session 6
- Starting the Session 6
- Windows File Association 6
- Exiting the Session 7
- Resetting the Session 7

Execution Command Syntax 7

Initialization Files 10

Environment Variables 10

Command Syntax 11

Command Line Help 12

Environment Commands 13
- Working Directory 13
- File Manipulation 13
- CPU Time 13
- Comment 13
- History 14
- Error Logging 14
- Determining the CUBIT Version 14
- Echoing Commands 14
- Digits Displayed 14

Cubit Memory 15
- Data Types 15
- Maximum Memory 15

Saving and Restoring a Cubit Session 15
- CUBIT File Method 15

Interrupting Running Tasks 17
Graphical User Interface 17
CUBIT Application Window 17
Customizing the Application Window 18
Control Panel 21
Control Panel Functionality 21
ID Input Entry Methods 23
Right-Click Context Menu for ID Input Fields 24
Value Fields 24
Control Panel Overview 24
Graphics Window 31
View Navigation in the GUI 31
Rotations 32
Zooming 34
Panning 35
Selecting Entities in the GUI 36
Key Press Commands for the GUI 37
Changing the Display Mode in the GUI 38
Right Click Commands for the GUI Graphics Window 39
With Entity Selected 39
Without Entity Selected 39
Tree View 39
Geometry Tree 40
Drag and Drop 41
Right-Click Menu Functions 41
Geometry Power Tools 42
Geometry Analysis Tools 43
Geometry Repair Tools 44
Right Click Menu 46
Meshing Tools 46
Steps to Using Meshing Power Tool 46
Right Click Context Menu 47
Mesh Quality Tools 47
Property Editor 48
Editing Entity Attributes from the Property Editor 49
Command Line Workspace 51
Command Window 51
Entering Commands 51
Repeating Commands 51
Error Window 52
Script Window 52
Docking and Undocking the Input Window 52
Journal File Editor 52
Journal Editor Toolbar 53
Right Click Menu 54
Toolbars 54
File 54
Display 54
Select 55

Drop Down Menus 55
File 55
Edit 56
View 56
Display 56
Tools 56
Help 56
Importing Files from the GUI 56
Importing ACIS/IGES/STEP Files 57
Importing Exodus II Files 57
1. Importing Mesh Geometry 57
2. Importing a Free Mesh 57
Importing Facet/AVS/STL Files 58
Importing CUB Files 58
Options Menu 58
Display Preferences 58
General Preferences 59
Geometry Defaults 59
History Preferences 59
Layout Preferences 59
Buttons 60
Cubit Layout Settings 60
Mesh Defaults 60
Mouse Settings 60
Command Recording and Playback 60
Journal File Creation and Playback 60
Recording a Session 60
Replaying a Session 60
Controlling Playback of Journal Files 61
Command Line Behaviors 61
Automatic Journal File Creation 62
Controlling Automatic Journal File Creation 62
Recording Graphics Commands 62
Recording Entity IDs and Names 62
Recording APREPRO Commands 62
Recording Errors 62
Idless Journal Files 63
Graphics Window Control 63
Updating the Display 64
Prevent Graphics From Updating 64
Command Line View Navigation: Zoom, Pan and Rotate 65
Rotation 65
Panning 65
Zooming 65
Graphics Modes 66
Truehiddenline Options 67
Displaying Using the Element Facets 67
Drawing and Highlighting Entities 68
Drawing Other Objects 68
Displaying Entity Orientation 68
Volume Sources and Targets 69
Model Axis 69
Surface Isoparameter Lines 69

Geometry Preview 69

Mesh Slicing 69
Notes on Mesh Slicing 70
Mesh Slicing Command 70

Entity Labels 70

Colors 71
Color Definitions 71
Specifying Colors in Commands 72
Assigning Colors 72

Geometry and Mesh Entity Visibility 73

Graphics Camera 73
Changing Camera Attributes Directly 74

Graphics Lighting Model 75

Graphics Window Size and Position 75
Using Multiple Windows 75

Saving Graphics Views 76

Hardcopy Output 76
Screen Capture Programs 77

Miscellaneous Graphics Options 77
Silhouette Lines 77
LineWidth 77
Highlight LineWidth 77
Text Size 78
Point Size 78
Graphics Status 78
Graphics Scale 78
Model Axis 78
Corner Axis (Triad) 78
Resetting the Graphics 78
Shrink 79

Entity Selection and Filtering 80

Command Line Entity Specification 80
Types of Entity Range Input 81

Extended Command Line Entity Specification 82
Extended Parsing Syntax 82
Keywords 82
Functions 83
Precedence 83

Selecting Entities with the Mouse 83
Entity Selection 85
Query Selection 86
Multiple Selected Entities 86
Information About the Selection 86
Picked Group 86
Substituting Selection into Other Commands 86

Entity Filtering 87
Executing Filters 87
Quality Filter Commands 88
Execute Filter Commands 88
Registering Filters 91
Filtering with Registered Filters 92
Filter Operations 92
List 92
Suppress/Resume 93
Move 93
Delete 93
Register Filter Commands 93
Location, Direction and Axis Specification 96
Specifying a Location 97
Position (XYZ values) 97
Last Location Used in a Command 97
Node or Vertex 97
On a Curve 98
On a Surface 98
Center 98
Extrema 98
Between 98
Move 99
Swing 99
Multiple Location Specification 99
Previewing a Location 99
Specifying a Location on a Curve or Curves 100
Start, Midpoint, or End 100
Fraction 100
Distance 100
{Close_To|At} Location 100
Extrema 101
Segment 101
Crossing 101
Previewing a Location on a Curve 101
Specifying a Direction 101
Vector (XYZ values) 102
Last Direction Used 102
X|Y|Z|Nx|Ny|Nz 102
On Curve Tangent 102
On Surface Normal 103
From Location 103
Rotate 103
Cross 104
Reverse 104
Previewing a Direction 104
Specifying an Axis 104
Last 104
Specify an origin and a vector 104
Revolve an axis about an axis 105
Previewing an Axis 105

Drawing a Location, Direction, or Axis 105
Listing Information 106
List Model Summary 106
List Geometry 107
List Mesh 108
List Special Entities 108
List Cubit Environment 109
Message Output Settings 109
Graphical Display Information 111
Memory Usage Information 111
Measuring Distances Between Entities 111
Measure Between 111
Measure Small 111

Geometry 112
CUBIT Geometry Formats 112
Setting the Geometry Kernel 112
Terms 112
Topology 113
Bodies and Volumes 113
Non-Manifold Topology 113
ACIS Geometry Kernel 113
Granite Geometry Kernel 113
Limitations 113
Mesh-Based Geometry 115
Creating Mesh-Based Geometry Models 115
Improving Mesh-Based Geometry Models for Meshing 116
Meshing Mesh-Based Models 117
Exporting Mesh-Based Geometry 117
Geometry Creation 117
Bottom-Up Geometry Creation 118
Creating Vertices 118
Creating Curves 119
Creating Surfaces 122
Creating Bodies 126
Geometric Primitives 130
General Notes 132

Creating Bricks 132
Creating Cylinders 132
Creating Prisms 133
Creating Frustrums 133
Creating Pyramids 133
Creating Spheres 134
Creating Toruses 134
Geometry Transforms 134
Align Command 135
Copy Command 135
Move Command 135
Moving Other Geometric Entities 136
Moving Bodies Relative to Other Geometric Entities 136
Moving Merged Entities 136
Move Undo 136
Scale Command 136
Rotate Command 137
Reflect Command 137
Geometry Booleans 137
Intersect 137
Subtract 138
Unite 138
Geometry Cleanup and Defeaturing 138
Healing 138
Analyzing Geometry 139
Healer Settings 139
Healing Attributes 140
Auto Healing 140
Spline Removal 141
What if Healing is Unsuccessful? 141
Regularizing Geometry 141
Finding Surface Overlap 142
Facetted Representation 142
Find Overlap Settings 142
Validating Geometry 143
Debugging Geometry 144
Geometry Accuracy 145
Tweaking Geometry 145
Tweaking Vertices 145
Tweaking a Vertex With a Chamfer 145
Tweaking a Vertex With a Non-Equal Chamfer 146
Tweaking a Vertex With a Fillet Radius 146
Tweaking Curves 147
Tweaking Surfaces 149
Tweaking a Surface Using an Offset 149
Tweaking a Surface by Moving 150

Tweaking a Surface by Replacement 150
Tweaking a Conical Surface 151

Removing Geometric Features 152

Removing Vertices 152

Removing Surfaces 152
Remove Sliver Surface 153

Trimming and Extending Curves 153
Trimming a Curve 154
Extending a Curve 155

Geometry Decomposition 155

Web Cutting 155
General Notes 156

Chop Command 156

Web Cutting with a Planar or Cylindrical Surface 156
Coordinate Plane 157
Planar Surface 157
Plane from 3 Points 157
Plane Normal to Curve 157
Cylindrical Surface 157
Previewing a Web Cut Plane 158

Web Cutting with an Arbitrary Surface 158

Web Cutting using a Tool or Sheet Body 158

Web Cutting by Sweeping Curves or Surfaces 159
Webcut by sweeping a surface along a trajectory 159
Webcut by sweeping a surface about an axis 160
Webcut by sweeping a curve(s) along a trajectory 161
Webcut by sweeping a curve(s) about an axis 161

Web Cutting Options 161

Web Cutting Preview 161
Preview a Webcutting Plane 161
Preview a Web Cutting Cylinder 162

Splitting Geometry 162

Split Curve 162

Split Surface 162
Split Across 163
Split (Automatically) 164

Split Periodic Surfaces 175

Separating Multi-Volume Bodies 175

Section Command 175

Geometry Imprinting and Merging 175

Imprinting Geometry 176

Merging Geometry 176
Merge geometry automatically 177
Test for merging in a specified group of geometry 177
Force merge specified geometry entities 177
Preventing geometry from merging 177

Examining Merged Entities 177
Merge Tolerance 178
Unmerging 178
Using Geometry Merging to Verify Geometry 179
Virtual Geometry 179
Composite Geometry 179
Composite Curves 180
Composite Surfaces 180
Controlling the Surface Evaluation Method for Composite Surfaces 180
Composite Determination 181
Partitioned Geometry 181
Partitioned Curves 182
Partitioned Surfaces 182
Partitioning with Vertices and Nodes 182
Partitioning with Hard Points 182
Partitioning with Polylines 182
Partitioning with Curves 183
Partitioning with Mesh Edges 183
Partitioning with Faces or Triangles 183
Partitioned Volumes 184
Using Mesh Intersections to Partition Surfaces 184
Removing Partitions 186
Collapse Angle 186
Deleting Virtual Geometry 188
Removing Virtual Geometry 188
Using The Delete Command With Composites 189
Using the Delete Command With Partitions 189
Geometry Orientation 189
Adjusting Orientation 189
Geometry Groups 190
Basic Group Operations 190
Geometry Groups 190
Mesh Groups 191
Deleting Groups 191
Cleaning Out Groups 191
Groups in Graphics 191
Propagated Hex Groups 191
Propagated Hex Group Starting on a Face 192
Ending at a Surface 192
Ending at a Face 192
Number of Times 192
Ending at a Surface with Multiple 193
Ending at a Face with Multiple 193
Number of Times with Multiple 193
Ending at Face with Direction 193
Ending at Surface with Direction 193
Number of Times with Direction 194
Propagated Hex Group Starting on a Surface 194
Ending at a Surface 194
Number of Times 194

Ending at a Surface with Multiple 195
Number of Times with Multiple 195
Ending at Surface with Direction 195
Number of Times with Direction 195

Naming Convention for Propagated Hex Groups 196

Quality Groups 197

Geometry Attributes 197

Entity Names 197
Automatic Name Creation 197
Automatic Name Propagation 198

Entity IDs 198
Gaps in ID space 198

Persistent Attributes 199

Attribute Behavior 199

Attribute Types 199

Attribute Commands 200
Control By Attribute Type or Geometric Entity 200

Using CUBIT Attributes 201

Geometry Metadata 201
Assembly Structure 201
Viewing Metadata 202
Modifying Metadata 202
Viewing and Modifying Global Metadata 202
Removing Metadata Associations 202
Replacing Metadata on Imported Geometry 202
Exporting Metadata 203

Geometry Deletion 203

Mesh Generation 204
Element Types 204
Mesh Generation Process 204

Interval Assignment 205

Interval Firmness 205
Precedence 206

Explicit Specification of Intervals 206

Automatic Specification of Intervals 206
Default auto interval specification 208
Maximum Spanning Angle on Arcs 208

Interval Matching 209

Periodic Intervals 210

Relative Intervals 210

Mesh Interval Preview 211

Meshing Schemes 211
Traditional Meshing Schemes 211
Free Meshing Schemes 211
Conversional Meshing Schemes 211
Duplication Meshing Schemes 212
General Meshing Information 212

Bias Dualbias 213
Circle 214
Curvature 215
Equal 215
Hole 215
Mapping 216
Pave 217
Element Shape Improvement 218
Controlling Flattening of Elements 218
Controlling the Grid Search for Intersection Checking 218
Controlling the Paver Sizing Function 218
Surface Vertex Types 219
Surface Vertex Commands 219
Listing and Drawing Vertex Types 220
Triangle Vertex Types 220
Adjusting the Automatic Vertex Type Selection Algorithm 220
Volume Curve Types 221
Pentagon 221
Pinpoint 222
Polyhedron 222
Sphere 224
STransition 225
Submap 227
Stretch 228
Sweep 229
Multisweep 231
Smoothing Swept Meshes 232
Some helpful hints in using sweep 233
Autosmooth 234
Grouping Sweepable Volumes 234
TetMesh 234
Using tets as the basis of an unstructured hexahedral mesh 235
Conforming the tetmesh to internal features 235
Tetprimitive 236
TriDelaunay 237
TriMap 237
TriMesh, TriAdvance 238
TriPave 239
TriPrimitive 240
Radialmesh 240
Dice 245
Refining a Mesh with Dicing 245
Detailed Discussion: 245
Extended Dicing Commands 246
Additional Dicing Commands 247
Constraining Nodes to Geometry: 247
Deleting a Fine Mesh 247
Interaction with Dicer Sheets 247

HTet 248
Unstructured 248
Structured 248
QTri 249
THex 250
Copying a Mesh 251
Mirroring a Mesh 252
Automatic Scheme Selection 254
Default Scheme Selection 254
Auto Scheme Selection General Notes 254
Scheme Firmness 255
Surface Auto Scheme Selection 255
Volume Auto Scheme Selection 256
Meshing the Geometry 256
Default Scheme and Interval Selection 256
Remeshing a Volume 257
Remeshing a Swept Volume Mesh 257
Continuing Meshing After a Mesh Failure 257
Mesh Quality Assessment 257
Metrics for Triangular Elements 258
Approximate Triangular Quality Definitions: 258
Comments on Algebraic Quality Measures 258
References for Triangular Quality Measures 259
Metrics for Quadrilateral Elements 259
Quadrilateral Quality Definitions 260
Comments on Algebraic Quality Measures 260
References for Quadrilateral Quality Measures 261
Details on Robinson Metrics for Quadrilaterals 261
Metrics for Tetrahedral Elements 261
Tetrahedral Quality Definitions 262
References for Tetrahedral Quality Measures 262
Metrics for Hexahedral Elements 262
Hexahedral Quality Definitions 263
References for Hexahedral Quality Measures 264
Mesh Quality Command Syntax 264
Quality Options 264
Scope 264
Draw 264
List 265
Filter 265
Mesh Quality Example Output 265
Automatic Mesh Quality Assessment 267
Controlling Mesh Quality 268
Skew Control 268
Propagate Curve Bias 268
Adjust Boundary 268

Coincident Node Check 269
Mesh Modification 269
Mesh Smoothing 269
Centroid Area Pull 271
Equipotential 271
Laplacian 271
Condition Number 272
Mean Ratio 273
Winslow 273
Mesh Refinement 274
Uniform Mesh Refinement 274
Refining at a Geometric or Mesh Feature 275
Hexahedral Refinement Using Sheet Insertion 276
Refining at a Geometric Feature 277
Refinement at a Boundary Surface 277
Refining along a path 278
Refining a Hex Sheet 278
Hex Sheet Drawing 281
Mesh Coarsening 281
Triangle Coarsening 281
Hexahedral Coarsening 283
Extracting a Single Hex Sheet 283
Extracting multiple sheets along a curve 283
Uniform hex coarsening 284
Mesh Cutting 284
Coordinate Plane 285
Planar Surface 285
Plane from 3 points 285
Extended Surface 285
Meshcut Options 285
Meshcutting Scope 285
Meshcutting Example 286
Collapsing Mesh Edges 290
Mesh Grafting 291
Grafting Options 291
Grafting Scope 291
Node and Nodeset Repositioning 294
Mesh Validity 295
Mesh Adaptivity and Sizing Functions 295
Adaptive Curve Meshing 295
Adaptive Surface Meshing 295
Adaptive Volume Meshing 296
Geometry Adaptive Sizing Function (Skeleton Sizing) 297
Command Line Syntax 298
Basic Arguments 298
Advanced Arguments 299
Lattice Arguments: 299
Source Entity Arguments 299
Interpolation Arguments 299
Skeleton with Other Sizing Controls 299
Limitations 300

Bias Sizing Function 300
Constant Sizing Function 305
Curvature Sizing Function 305
Linear Sizing Function 307
Interval Sizing Function 308
Inverse Sizing Function 309
Exodus II-based Field Function 310
Curve Meshing with Exodus II - based Field Functions 311
Mesh Deletion 311

Importing and Exporting Files 312

Importing Geometry 312
Other Geometry Formats 312
Importing ACIS Files 312
Importing ACIS files at startup 313
Importing FASTQ Files 313
Importing STEP Files 314
Exporting a STEP file from Pro/Engineer 314
Setting Up CUBIT to Use STEP Tools 315
Importing IGES Files 315
Manifold Solid B-rep Objects (MSBO) 316
Importing Facet Files 316
Facet File Format 317
Feature Angle 317
Smooth Curves and Surfaces 317
Merge 318
Make elements 318
Stitch 318
Improve 318
Importing Granite Files 319
Exporting Geometry 319
Exporting ACIS Files 319
Exporting STEP Files 320
Exporting IGES Files 320
Exporting Granite Files 320
Exporting Facet Files 321
Importing a Mesh 321
Importing 2D Exodus Files 321
Importing Exodus II Files 322
Mesh-Based Geometry 322
File Name 322
Blocks 322
Nodesets/Sidesets 323
Feature Angle 324
Smooth Curves and Surfaces 324
Apply Deformations 325
Merge 325
Merge Nodes 325

Export Facets 325

Importing a Free Mesh 326
Importing a Mesh with Nodeset Associativity 326
Importing a Mesh onto Modified Geometry 327
Importing a Mesh without Geometry Associativity 327
Specifying a Portion of the Mesh to be Imported 327
Unique Genesis IDs and Shell Options 327

Exporting the Finite Element Model 327
Other Formats 327

Exporting an Exodus II File 328
Converting an Exodus II file to ASCII 328

Exporting ABAQUS Files 329

Finite Element Model 330

Finite Element Model Definitions 330
Element Blocks 330
Nodesets 330
Sidesets 330
Element Types 331

Element Block Specification 331
Creating Element Blocks 332
Default Element Blocks 332
Defining the Element Type 332
Assigning Attributes to Blocks 333
Displaying Element Blocks 334
Deleting Element Blocks 334
Automatically Assigning Mesh Edges to a Block (Rebar) 334
2D Elements 334

Nodeset and Sideset Specification 335
Creating Nodesets and Sidesets 335
Grouping Faces on a Surface into a Sideset 336
Deleting Nodesets and Sidesets 336
Displaying Nodesets and Sidesets 336
Nodeset Associativity Data 337

Exodus II Model Title 337

Transforming Mesh Coordinates 338

Exodus Coordinate Frames 338

Exodus II File Specification 339
Exodus II Manual 339
Element Block Definition Examples 339
Multiple Element Blocks 339
Surface Mesh Only 339
Two-dimensional Mesh 340

Step-By-Step Tutorials 341
Additional Tutorials 341

Command Line Basic Tutorial 342
Overview 342

Command Line Basic Tutorial 343
Step 1: Beginning Execution 343

Command Line Basic Tutorial **344**
Step 2: Beginning Execution 344
Command Line Basic Tutorial **345**
Step 3: Creating the Cylinder 345
Command Line Basic Tutorial **346**
Step 4: Adjusting the Graphics Display 346
Command Line 346
Mouse 346
Command Line Basic Tutorial **347**
Step 5: Forming the Hole 347
Command Line Basic Tutorial **348**
Step 6: Setting Interval Sizes 348
Command Line Basic Tutorial **349**
Step 7: Surface Meshing 349
Command Line Basic Tutorial **350**
Step 8: Surface Meshing 350
Command Line Basic Tutorial **352**
Step 9: Inspecting the Model 352
Command Line Basic Tutorial **355**
Step 10: Defining Boundary Conditions 355
Command Line Basic Tutorial **355**
Step 11: Exporting the Mesh 355
GUI Basic Tutorial **355**
Overview 355
GUI Basic Tutorial **357**
Step 1: Beginning Execution 357
GUI Basic Tutorial **358**
Step 2: Creating the Brick 358
GUI Basic Tutorial **360**
Step 3: Creating the Cylinder 360
GUI Basic Tutorial **361**
Step 4: Adjusting the Graphics Display 361
GUI Basic Tutorial **362**
Step 5: Forming the Hole 362
GUI Basic Tutorial **364**
Step 6: Setting Interval Sizes 364
GUI Basic Tutorial **367**
Step 7: Surface Meshing 367
GUI Basic Tutorial **370**
Step 8: Volume Meshing 370
GUI Basic Tutorial **372**
Step 9: Inspecting the Model 372
GUI Basic Tutorial **374**
Step 10: Defining Boundary Conditions 374
GUI Basic Tutorial **376**
Step 11: Exporting the Mesh 376
Power Tools GUI Tutorial **377**
Overview 377

Power Tools GUI Tutorial 378
Step 1: Import the Geometry 378
Power Tools GUI Tutorial 381
Step 2: Analyze the Geometry 381
Power Tools GUI Tutorial 385
Step 3: Healing the Geometry 385
Power Tools GUI Tutorial 388
Step 4: Mesh Power Tools 388
Power Tools GUI Tutorial 390
Step 5: Splitting Filleted Surfaces 390
Power Tools GUI Tutorial 394
Step 6: Web Cutting 394
Power Tools GUI Tutorial 401
Step 7: Removing Small Surfaces 401
Power Tools GUI Tutorial 407
Step 8: Tweaking Surfaces 407
Power Tools GUI Tutorial 410
Step 9: Imprint/Merge 410
Power Tools GUI Tutorial 412
Step 10: Compositing Surfaces 412
Power Tools GUI Tutorial 420
Step 11: Meshing the Model 420

Appendix 427
Examples 427
General Comments 427
Example:Simple Internal Geometry Generation 428
Meshing with Autoscheme 429
Example: Octant of a Sphere 429
Example: Box Beam 430
Block, Block Attribute 432
NodeSet Move 432
Merge 432
Example: Thunderbird 432
Example: Advanced Tutorial 435
Alpha Commands 436
Automatic Detail Suppression 436
Example 437

Automatic Geometry Decomposition 438
FeatureSize 438
HexTet 439
HexToVoid 440
Optimize Jacobian 440
Plastering 441
Randomize 441
Sculpting 442
Stride 443
Super Sizing Function 443
Test Sizing Function 444
Transition 445
Untangle 447
Whisker Weave 448
Whisker Weaving Basic Commands 449
Whisker Weaving Options 450
Available Colors 450
Element Numbering 454
Node Numbering 454
Side Numbering 454
Triangular Shell Element Numbering 455
Node Ordering 455
Side Set Side Ordering 455
Mesh Editing 456
Model Acquisition 456
Manipulating Topology and Connectivity: Imprinting and Merging and Unmerging 456
Options for mesh transfer 458
Geometry (and Mesh) Manipulation 458
Mesh Manipulation 458
FullHex vs. NodeHex Representation 458
APREPRO 458
APREPRO Syntax 459
APREPRO Rules 459
1. Functions 459
2. Variables 459
3. Numbers 459
4. Strings 459
5. Operators 459
6. Delimiters 460
7. Expressions 460
8. Algebraic Expressions 460
9. String Expressions 460
10. Relational Expressions 460
11. Conditional Expressions 460
APREPRO Operators 460
1. Arithmetic Operators 461
2. Assignment Operators 461
3. Relational Operators 462
4. Boolean Operators 462
5. String Operators 463

APREPRO Predefined Variables 463
APREPRO Functions 464
1. Mathematical Functions 464
2. CUBIT Functions 466
3.String Functions 467
APREPRO Additional Functionality 469
1. File Inclusion 469
2. Conditionals 469
3. Loops 470
APREPRO Journaling 470
APREPRO Comments 470
Significant Figures 470
Periodic Space Filling Models (Tile) 471
Initial setup 471
Creating Nodesets 471
Smoothing 471
Example 472
Troubleshooting Guide 472
References 474
Credits 476
Quick Reference 478

Introduction

Key Features

How to Use This Manual

Licensing, Distribution, and Installation

Hardware Requirements

Trademark Notice

Welcome to CUBIT, the Sandia National Laboratory automated mesh generation toolkit. CUBIT is an full-featured software toolkit for robust generation of two- and three-dimensional finite element meshes (grids) and geometry preparation. Its main goal is to reduce the time to generate meshes, particularly large hex meshes of complicated, interlocking assemblies. It is a solid-modeler based preprocessor that meshes volumes and surfaces for finite element analysis. Mesh generation algorithms include quadrilateral and triangular paving, 2D and 3D mapping, hex sweeping and multi-sweeping, tet meshing, and various special purpose primitives. CUBIT contains many algorithms for controlling and automating much of the meshing process, such as automatic scheme selection, interval matching, sweep grouping and sweep verification, and also includes state-of-the-art smoothing algorithms

The CUBIT environment is designed to provide the user with a powerful toolkit of meshing algorithms that require varying degrees of input to produce a complete finite element model. Many CUBIT users want to experiment with capabilities as soon as possible. Hence, CUBIT releases often contain algorithms which are not quite ready for production use.

The overall goal of the CUBIT project is to reduce the time it takes a person to generate an analysis model. Generating meshes for complex, solid model-based geometries requires a variety of tools. Many CUBIT tools are completely automatic, while others require user input. Usually, the automatic choices can be over-ridden by the user if necessary. Most meshing capabilities are integrated into the common CUBIT framework; There are also stand-alone tools like Verde. The user is encouraged to become familiar with all of the available tools, so that he can choose the right one for the job.

Key Features

Geometry Creation, Modification, and Healing

CUBIT usually relies on the ACIS solid modeling kernel for geometry representation; there is also mesh-based geometry, and a Granite port for Pro Engineer files. Other solid model kernels are planned. Geometry is imported or created within CUBIT. Geometry is created bottom-up or through primitives. CUBIT imports ACIS SAT files. CUBIT can also read STEP, IGES, and FASTQ files and convert them to the ACIS kernel. SolidWorks, AutoCAD, and some other commercial CAD systems can write SAT files directly.

Once in CUBIT, an ACIS model is modified through Booleans. Without changing the geometric definition of the model, the topology of the model may be changed using virtual geometry. For example, virtual geometry can be used to composite two surfaces together, erasing the curve dividing them.

Sometimes, an ACIS model is poorly defined. This often happens with translated models. The model can be healed inside CUBIT.

Non-Manifold Topology

Typical assembly meshes require contiguous mesh across multiple parts in an assembly. CUBIT accomplishes this by taking the two touching surfaces of neighboring volumes, and merging them into a single surface. There will be only one mesh of the surface, and both volume meshes will share that surface mesh. (In contrast, some meshing packages keep two surfaces, and take steps to ensure their mesh connectivity and positions match.)

These shared surfaces are called "non-manifold topology". Geometric models are usually imported into CUBIT as manifold (non-shared) models; then, surfaces which pass a geometric and topological comparison are "merged". A similar technique is used to merge model edges and vertices across parts. These comparisons are performed automatically, and can optionally be restricted to subsets of the model (to allow representations of such features as slide lines).

Geometry Decomposition

Solid models often require decomposition to make them amenable to hexahedral meshing. CUBIT contains a wide variety of tools for interactive geometry decomposition, and a capability for performing automatic geometry decomposition is also under development.

Mesh Generation

CUBIT contains a variety of tools for generating meshes in one, two and three dimensions. While the primary focus of CUBIT is on generating unstructured quadrilateral and hexahedral meshes, algorithms are also available for structured mesh generation and triangle/tetrahedral mesh generation. Several algorithms for generating mixed hex-tet meshes are also being developed.

Boundary Conditions

CUBIT uses the EXODUS-II format for importing and exporting mesh data. EXODUS represents boundary conditions on meshes using Element Blocks, Nodesets, and Sidesets. Element Blocks are used to group elements by material type. Nodesets are used to group nodes. Other tools can apply nodal boundary conditions to these sets, for example enforced displacement or nodal temperature values. Sidesets are used to group sides of elements, such as faces of hexes or edges of quads. Other tools can apply face-based and edge-based boundary conditions to these sets, for example pressure or heat flux.

Using Element Blocks, Nodesets and Sidesets, a mesh and boundary conditions can be specified in an analysis-independent manner. Typically this specification is combined with an additional data file which designates the specific type of boundary condition (temperature, displacement, pressure, etc.), along with boundary condition values.

Element Types

CUBIT supports a wide variety of element types, including 1d, 2d, and 3d elements of various orders. Each block has a unique element type. The element type is specified after the block is created, and after mesh generation (recommended). Higher order nodes are generated when the element type is specified. Higher order nodes are projected to curved geometry, depending on the user-settable node constraint flag.

Graphics Display Capabilities

CUBIT uses the VTK package for its graphics and rendering engine. CUBIT can display geometric and mesh entities in several modes, including hidden line, shaded, transparent or wireframe modes. CUBIT supports screen picking of geometric and mesh entities, as well as mouse-controlled view transformations like rotate, pan, and zoom. VTK takes advantage of hardware acceleration on most supported platforms. Image files of any displayed image can also be generated. CUBIT can also be run without graphics, to allow execution in batch mode or over slow network connections.

Graphical User Interface

A full graphical user interface (GUI) with the standard look and feel consistent with major platforms is available on all supported Cubit platforms. The GUI version can improve productivity, making new users aware of the wide range of CUBIT capabilities, and freeing new and experienced users from having to remember esoteric syntax. The GUI and non-GUI versions create and play back identical journal files, making it easier to switch from one environment to the other.

Command Line Interface

In the command line interface, commands are specified by text rather than mouse clicks. Commands can be entered interactively or in batch mode by playing back a journal file. The command line interface is available in the GUI through a window. The non-GUI version supports graphical picking and echoing to the command line, and also mouse-driven view transformations, but no menus and dialog boxes. The command line and GUI dialog boxes support the APREPRO preprocessor, which allows parameterization of input. The non-GUI version is available on all platforms, including Windows.

How to Use This Manual

This manual provides specific information about the commands and features of CUBIT. It is divided into chapters, which roughly follow the process in which a finite element model is created, from geometry creation to mesh generation to boundary condition application. An example is provided in a tutorial. Appendices contain advanced topics, advanced examples, installation instructions, a troubleshooting guide, and references.

Integrated in CUBIT are algorithms and tools which are in a user-beware state. As they are further tested (often with the assistance of users) and improved, the tool becomes more stable and production-worthy. Since documentation of the tool is necessary for actual use, we have included the documentation of all available tools. However, a "hammer" icon is placed next to some capabilities as a warning.

Certain portions of this manual contain information that is vital for understanding and effectively using CUBIT. These portions are highlighted with a "key" icon.

Licensing, Distribution and Installation

The CUBIT code is available for use by personnel inside Sandia, any other government laboratory, or to personnel performing work under contract by a US government entity. In addition, CUBIT can be licensed for non-commercial and research use. For more information on licensing of CUBIT, see the CUBIT web page (http://malla.sandia.gov/cubit/index.html) or send email to cubit-dev@sandia.gov.

CUBIT installations have use restrictions. THE CUBIT CODE CANNOT BE COPIED TO ANOTHER COMPUTER AND THE NUMBER OF USER SEATS ON EACH COMPUTER OR LAN IS LIMITED. If additional user seats or additional copies of CUBIT are required, you MUST contact us to acquire them.

CUBIT incorporates code modules developed by outside code vendors and licensed to the CUBIT project. Since the number of licenses for these modules is limited, CUBIT cannot be copied and redistributed without notifying the CUBIT team.

CUBIT is distributed in statically linked executable form for each supported platform. Supported platforms include the HP 9000 series running HP-UX, Sun SPARCstations running Solaris, Linux, SGI machines running IRIX, Mac X version 10.3 and higher (Cubit running on the Mac is considered beta) and Windows NT/2000 machines. Additional platforms will be added as required.

Instructions for obtaining the CUBIT code will be given after licensing arrangements have been completed.

In addition to the CUBIT executable, the suite of example problems described in this manual is available upon request.

Hardware Requirements

Cubit is available on the following platforms:

- Linux RedHat 9.0
- SGI Irix 6.5 32- and 64-bit
- Sun Solaris 8
- HP-UX 11.x
- Windows 2000/XP
- Mac OS X

The Graphical User Interface version is available on all platforms.

Trademark Notice

HP-UX is a registered trademark of Hewlett-Packard Company.

Sun, SunOS, and Solaris are registered trademarks of Sun Microsystems, Inc.

IRIX is a registered trademark of Silicon Graphics, Inc.

ACIS is a proprietary format developed by Spatial Technologies.

Granite is a proprietary format developed by Parametric Technology Corporation

All other trademarks are the property of their respective owners.

CUBIT Mailing Lists

The CUBIT team maintains a couple of mailing lists to help our users.

1) The cubit-announce mailing list is a very low-volume mailing list intended to provide news of new releases and other items of major importance. To subscribe to this list, send a message to: majordomo@sandia.gov
with the body of the message being:

subscribe cubit-announce

2) The cubit users mailing list is a medium-volume mailing list intended for our users to communicate with each other and ask help of the user community. It also contains the same announcements as the cubit-announce mailing list. To send questions or comments to this list, send email to:

cubit@sandia.gov

Users can subscribe to the cubit mailing list by emailing majordomo@scico.sandia.gov with a message body consisting of the single line:

subscribe cubit

An additional mailing list, cubit-help@sandia.gov, has been created for direct communication with the CUBIT developers. These messages won't reach other users. This list should be used for topics that are not of general interest to others, including some bugs.

Note: The recommended use of an electronic mailing list to report bugs and request enhancements is not intended to discourage face-to-face discussion with CUBIT developers, but rather to minimize response time. Users are encouraged to discuss bugs, enhancements or general meshing issues with the CUBIT production meshing and development teams.

Environment Control

- Session Control
- Graphical User Interface
- Command Recording and Playback
- Graphics Window Control
- Entity Selection and Filtering
- Location, Direction, and Axis Specification
- Listing Information

The CUBIT user interface is designed to fill multiple meshing needs throughout the design to analysis process. The user interface options include a full graphical user interface, a modern command line interface as well as no-graphics and batch mode operation. This chapter covers the interface options as well as the use of journal files, control of the graphics, a description of methods for obtaining model information, and an overview of the help facility.

Session Control

- Starting and Exiting a CUBIT Session
- Execution Command Syntax
- Initialization Files
- Environment Variables
- Command Syntax
- Command Line Help
- Environment Commands
- CUBIT Memory
- Saving and Restoring a CUBIT Session
- Interrupting Running Tasks

This section provides an overview to session control in CUBIT. This includes information on starting and exiting a CUBIT session, running CUBIT in batch mode, initialization files, how to enter commands, file manipulation, changing the working directory, memory manipulation and more. Much of your ability to use CUBIT effectively depends on mastery of concepts in this section. Even experienced users will find it useful to review this section periodically.

Starting and Exiting a CUBIT Session

The following commands are used to control CUBIT execution.

Starting the Session

The command line version of CUBIT can be started on UNIX machines by typing "**cubit**" at the command prompt from within the CUBIT directory. If you have not yet installed CUBIT, instructions for doing so can be found in Licensing, Distribution and Installation. A CUBIT console window will appear which tells the user which CUBIT version is being run and the most recent revision date. A graphics window will also appear unless you are running with the **-nographics** option. For a complete list of startup options see the Execution Command Syntax section of this document. CUBIT can also be run with initialization files or in batch mode.

Windows File Association

Windows users have the option to associate .cub, .sat, and .jou files with CUBIT. This means that double-clicking on one of these files will open it automatically in CUBIT. This option is available during the installation process

Entering Commands

Commands can be entered at the command prompt

Exiting the Session

The CUBIT session can be discontinued with either of the following commands

Exit

Quit

Resetting the Session

A reset of CUBIT will clear the CUBIT database of the current geometry and mesh model, allowing the user to begin a new session without exiting CUBIT. This is accomplished with the command

Reset [Genesis | Blocks | Nodesets | Sidesets]

A subset of portions of the CUBIT database to be reset can be designated using the qualifiers listed. Advanced options controlled with the Set command are not reset.

You can also reset the number of errors in the current Cubit session, using the command

Reset errors [value]

which will set the error count to the specified value, or zero if the value is left blank.

Execution Command Syntax

Execution command syntax options for the command line version of CUBIT are:

cubit
-help (Print this summary)
-Include <$val> (Default path to search for input files)
-input $val (Playback commands in file $val)
-solidmodel <$val> (Read .sat or .cub from file $val)
-fastq <$val> (Read FASTQ file $val)
-initfile <$val> (Read $val as initialization file instead of $HOME/.cubit)
-batch (Batch Mode - No Interactive Command Input)
-nographics (Do not display graphics windows)
-noinitfile (Do not read .cubit file)
-noecho (Do not echo commands to console)
-nojournal (Do not write journal file)
-nodeletions (Do not allow file deletions)
-journalfile <$val> (Name of journal file, will be overwritten)
-restore [$val] (Name of restore file (default = cubit_geom.save.sat))
-maxjournal [$val] (Maximum number of journal files to write)
-warning [$val] (Warning Messages On/Off)
-information [$val] (Informational Messages On/Off)
-debug <$val> (Set specified flags on, e.g. 1,3,7-9 enables 1,3,7,8,9))
-display <$val> (Specify display to be used for graphics window)
-driver <$val> (Specify the type of driver to be used for graphics display)
-nooverwritecheck (Do not perform file export overwrite check)
-variable=<value> (Assign an aprepro variable a value)

Each of these are optional. If specified, the quantities in square brackets, **[$val]**, are optional and the quantities in angle brackets, **<$val>**, are required.

Options are summarized in more detail below:

-help	Print a short usage summary of the command syntax to the terminal and exit.
-initfile <$val>	Use the file specified by **<$val>** as the initialization file instead of the default set of initialization files. See Initialization Files
-noinitfile	Do not read any initialization file. This overrides the default behavior described in Initialization Files
-solidmodel <$val>	Read the ACIS solid model geometry or .cub file information from the file specified by **<$val>** prior to prompting for interactive input.
-batch	Specify that there will be no interactive input in this execution of CUBIT. CUBIT will terminate after reading the initialization file, the geometry file, and the **input_file_list.**
-nographics	Run CUBIT without graphics. This is generally used with the -batch option or when running CUBIT over a line terminal.
-display	Sets the location where the CUBIT graphics system will be displayed, analogous to the **-display** environment variable for the X Windows system. Unix only.
-driver <type>	Sets the **<type>** of graphics display driver to be used. Available drivers depend on platform, hardware, and system installation. Typical drivers include *X11* and *OpenGL*.
-nojournal	Do not create a journal file for this execution of CUBIT. This option performs the same function as the Journal Off command. The default behavior is to create a new journal file for every execution of CUBIT.
-journalfile <file>	Write the journal entries to **<file>**. The file will be overwritten if it already exists.
-maxjournal <$val>	Only create a maximum of **<$val>** default journal files. Default journal files are of the form cubit#.jou where # is a number in the range 01 to 99.

-nodeletions	Turn off the ability to delete files with the delete file '<filename>' command.
-nooverwritecheck	Turn off the file overwrite check flag. Files that are written may then overwrite (erase) old files with the same name with no warning. This is typically useful when re-running journal files, in order to overwrite existing output files. See the **set File Overwrite Check [ON|off]** command.
-restore	Restore the specified filename (or "cubit_geom") mesh and ACIS files, e.g. cubit_geom.save.g and cubit_geom.save.sat.
-noecho	Do not echo commands to the console. This option performs the same function as the Echo Off command. The default behavior is to echo commands to the console.
-debug=<$val>	Set to "on" the debug message flags indicated by **<$val>**, where **<$val>** is a comma-separated list of integers or ranges of integers, e.g. 1,3,8-10.
-information={on|off}	Turn **{on|off}** the printing of information messages from CUBIT to the console.
-warning={on|off}	Turn **{on|off}** the printing of warning messages from CUBIT to the console.
-Include=<path>	Set the patch to search for journal files and other input files to be **<path>.** This is useful if you are executing a journal file from another directory and that journal file includes other files that exist in that directory also.
-fastq=<file>	Read the mesh and geometry definition data in the FASTQ file **<file>** and interpret the data as FASTQ commands. See Reference See T. D. Blacker, FASTQ Users Manual Version 1.2, SAND88-1326, Sandia National Laboratories, (1988). for a description of the FASTQ file format.
<input_file_list>	Input files to be read and executed by CUBIT. Files are processed in the order listed, and afterwards interactive command input can be entered (unless the -batch option is used.)

<variable=value>	APREPRO variable-value pairs to be used in the CUBIT session. Values can be either doubles or character type (character values must be surrounded by double quotes.), Command options can also be specified using the **CUBIT_OPT** environment variable. (See User Environment Settings.)

Initialization Files

CUBIT can execute commands on startup, before interactive command input, through initialization files. This is useful if the user frequently uses the same settings.

On Unix or Windows, the following files are played back in order, if they exist, at startup:

$HOMEDRIVE$HOMEPATH/.cubit
$HOME/.cubit
$(current working directory)/.cubit

Where **$(current working directory)** is determined by the program itself and words starting with '**$**' are environment variables.

If the **-initfile <filename>** option is used on the command that starts cubit, then the other init files are skipped and only the specified filename is played back.

The **$CUBIT_DIR** file is installation specific. The **$HOME** file is user specific. The **$PWD** file is run-specific, read when starting up cubit from a particular meshing problem's subdirectory.

These files are typically used to perform initialization commands that do not change from one execution to the next, such as turning off journal file output, specifying default mouse buttons, setting geometric and mesh entity colors, and setting the size of the graphics window.

Environment Variables

CUBIT can interpret the following environment variables. These settings are only applicable to the Command Line Version of CUBIT and do not apply to the Graphical User Interface. See also the CUBIT_STEP_PATH and CUBIT_IGES_PATH environment variables. See also the CUBIT_DIR, HOMEDRIVE and HOMEPATH settings.

DISPLAY	The graphics window or GUI will pop-up on the specified X-Window display. This is useful for running CUBIT across a network, or on a machine with more than one monitor. Unix only.
CUBIT_OPT	Execution command line parameter options. Any option that is valid from the command line may be used in this environment variable. See Execution Command Syntax.
CUBIT_Journal	Specifies path and name to use for journal file. The specified path may contain the following %-escape sequences: %a - abbreviated weekday name %A - full weekday name %b - abbreviated month name %B - full month name %d - date of the month [01,31] %H - hour (24-hour clock) [00,23] %I - hour (12-hour clock) [01,12] %j - day of the year [1,366]

	%m - month number [1,12] %M - minute [00,59] %n - replaced with the next available number between 01 and 99. %p - "a.m." or "p.m." %S - seconds [00,61] %u - weekday [1,7], 1 is Monday %U - week of year [00,53] %w - weekday [0,6], 0 is Sunday %y - year without century [00,99] %Y - year with century (e.g. 1999) %% - a '%' character The default value is "cubit%n.jou". This creates journal files in the current directory named "cubit00.jou", "cubit01.jou", "cubit02.jou", etc. To keep the same naming scheme but create the files the /tmp directory, set CUBIT_JOURNAL to "/tmp/cubit%n.jou" To create journal files in directories according to the day of the week, first create directories named "Mon", "Tues", etc. CUBIT will not create them for you. Next set CUBIT_JOURNAL to "%a/%n.jou". This will create journal files named "01.jou" through "99.jou" in the appropriate directory for the current day of the week.

Command Syntax

The execution of CUBIT is controlled either by entering commands from the command line or by reading them in from a journal file. Throughout this document, each function or process will have a description of the corresponding CUBIT command; in this section, general conventions for command syntax will be described. The user can obtain a quick guide to proper command format by issuing the <keyword> help command; see Command Line Help for details.

CUBIT commands are described in this manual and in the help output using the following conventions. An example of a typical CUBIT command is:

{volume_list} Scheme Project [Source {surface_list} Target {surface_list}]

The commands recognized by CUBIT are free-format and abide by the following syntax conventions.

1. Case is not significant.

2. The "#" character in any command line begins a comment. The "#" and any characters following it on the same line are ignored.

3. Commands may be abbreviated as long as enough characters are used to distinguish it from other commands.

4. The meaning and type of command parameters depend on the keyword. Some parameters used in CUBIT commands are:

Numeric: A numeric parameter may be a real number or an integer. A real number may be in any legal C or FORTRAN numeric format (for example, 1, 0.2, -1e-2). An integer parameter may be in any legal decimal integer format (for example, 1, 100, 1000, but not 1.5, 1.0, 0x1F).

String: A string parameter is a literal character string contained within single or double quotes. For example, 'This is a string' .

Filename: When a command requires a filename, the filename should be enclosed in single or double quotes. If no path is specified, the file is understood to be in the current working directory. After entering a portion of a filename, typing a '?' will complete the filename, or as much of the filename as possible if there is more than one possible match.

A filename parameter must specify a legal filename on the system on which CUBIT is running. The filename may be specified using either a relative path (**../cubit/mesh.jou**), a fully-qualified path (**/home/jdoe/cubit/mesh.jou**), or no path; in the latter case, the file must be in the working directory or in a directory specified using the -path option to CUBIT (see Executing CUBIT for details.) Environment variables and aliases may also be used in the filename specification; for example, the C-Shell shorthand of referring to a file relative to the user's login directory (**~jdoe/cubit/mesh.jou**) is valid.

Toggle: Some commands require a "toggle" keyword to enable or disable a setting or option. Valid toggle keywords are "**on**", "**yes**", and "**true**" to enable the option; and "**off**", "**no**", and "**false**" to disable the option.

5. Each command typically has either:

* an action keyword or "verb" followed by a variable number of parameters, for example

Mesh Volume 1

Here Mesh is the verb and Volume 1 is the parameter.

* or a selector keyword or "noun" followed by a name and value of an attribute of the entity indicated, for example

Volume 1 Scheme Project Source 1 Target 2

Here Volume 1 is the noun, Scheme is the attribute, and the remaining data are parameters to the Scheme keyword.

The notation conventions used in the command descriptions in this document are:

- The command will be shown in a format that **looks like this**:
- A word enclosed in angle brackets (**<parameter>**) signifies a user-specified parameter. The value can be an integer, a range of integers, a real number, a string, or a string denoting a filename or toggle. The valid value types should be evident from the command or the command description.
- A series of words delimited by a vertical bar (**choice1 | choice2 | choice3**) signifies a choice between the parameters listed.
- A word enclosed in square brackets (**[optional]**) signifies optional input which can be entered to modify the default behavior of the command.

Command Line Help

In addition to the documentation you are currently viewing, CUBIT can give help on command syntax from the command line. For help on a particular command or keyword, the user can simply type **help <keyword>** . In addition, if the user has typed part of a command and is uncertain of the syntax of the remainder of the command, they can type a question mark **?** and help will be printed for the sequence of keywords currently entered. It is important to note that if the user has typed the keywords out of order, then no help will be found. If the user is not sure of the correct order of the keywords, the ampersand **&** key will search on all occurrences of whatever keywords are entered, regardless of the order. The results of this type of command are shown in the following listing.

```
CUBIT> volume 3 label fish
Completing commands starting with: volume, label.
Help not found for the specified word order.
CUBIT> volume 3 label fish &
Help for words: volume & label
Label Volume [ on | off | name [only|id] | id | interval |
        size | scheme | merge | firmness ] CUBIT> label volume 3 fish ?
Completing commands starting with: label, volume.
Label Volume [on|off|name [only|ids]|ids|interval|size|scheme|merge|firmness]
```

Help on Volume & Label

Environment Commands

- Working Directory
- File Manipulation
- CPU Time
- Comment
- History
- Error Logging
- Determining the CUBIT Version
- Echoing Commands
- Digits Displayed

Working Directory

The working directory is the current directory where journal files are saved. To list the current directory type

pwd

The current path will be echoed to the screen. By default, the current directory is the directory from which CUBIT was launched. The command to change the current directory is

cd "<new_path>"

The new path may be an absolute reference, or relative to the current directory. The <TAB> key will complete unique file references.

File Manipulation

A helpful addition is the ability to do a directory listing of a directory. The command for this is

ls ['<file_name>']

or

dir ['<file_name>']

Note also that you can delete files from the command line. The command for this is

delete file ['<file_name>']

The file_name may include the wildcard character *, but not the wildcard character ?, since the ? is used for command completion.

CPU Time

At times it is important to see how much cpu time is being used by a command. One function available to do this is the timer command. The syntax for this command is:

Timer [start|stop]

The start option will start a CPU timer that will continue until the stop command is issued. The elapsed time will be printed out on the command line. If no arguments are given, the command will act like a toggle.

Comment

This keyword allows you to add comments without affecting the behavior of CUBIT.

Comment '<text_written_to_journal_file>'

History

This command allows you to display a listing of your previous commands.

history [number_of_lines]

For example, if you type history 10, the most recent 10 commands will be echoed to the input window.

Error Logging

[set] Logging Errors {off | on file '<filename>'}

This setting will allow users to echo error messages to a separate log file.

Determining the CUBIT Version

To determine information on version numbers, enter the command Version. This command reports the CUBIT version number, the date and time the executable was compiled, and the version numbers of the ACIS solid modeler and the VTK library linked into the executable. This information is useful when discussing available capabilities or software problems with CUBIT developers.

Echoing Commands

By default, commands entered by the user will be echoed to the terminal. The echo of commands is controlled with the command:

[set] echo {on | off}

Digits Displayed

CUBIT uses all available precision internally, but by default will only print out a certain number of digits in order for columns to line up nicely. The user can override that with the "set digits" command:

set Digits [<num_to_list=-1>]

If the digits are set to -1, then the default number of digits for pretty formatting are used. If the digits are set to a specific number, such as 15, more digits of accuracy can be displayed. This may be useful when checking the exact position and size of geometric features.

The number of digits used for listing positions, vectors and lengths can be listed using the following command:

List Digits

Examples:

CUBIT> set digits 6

Coordinates and lengths will be listed with up to 6 digits.

CUBIT> set digits 20

For this platform, max digits = 15. Coordinates and lengths will be listed with up to 15 digits.

CUBIT> set digits -1

To reset digits to default, use 'set digits -1'

The number of coordinate and length digits listed will vary depending on the context.

Cubit Memory

Data Types

Cubit has the ability to change its internal data representation in order to decrease the memory it uses during execution. This can be done with the command:

Volume <id_range> Data_Type { Native|Reduced|File }

Native is the default data type Cubit uses. It is the fastest, but it creates the largest memory footprint.

Reduced data type reduces the internal data but still keeps it in memory. The memory footprint is smaller than data type native.

File data type writes the data out to disk and retrieves it as necessary. Mesh of this data type uses the least memory but executes slowest.

Files can be correctly written to ExodusII format using the export genesis 'file.g' command regardless of the storage method for each volume. There are two cautions when using this command:

1. When in reduced/file mode, the mesh is largely invisible to graphics and inquiry functions.
2. To store higher order data, care must be used. The data must be made higher order before reducing to a reduced type. This usually involves explicitly creating a block to hold the volume(s) before meshing to get correct results. For example:

   block 1 vol 1
   block 1 element type HEX20
   mesh vol 1
   vol 1 data_type file

To set all to-be-meshed volumes to a desired data type,
so to avoid setting each one explicitly, use the command:

Volume Default Data_Type { Native|Reduced|File }

The meshed volumes will automatically be in the set data type.

Maximum Memory

Cubit allows the user to turn on automatic memory management for mesh storage with the command:

Set Maximum Memory [On|Off|Value(in MB)]

When the memory used by Cubit approaches the amount specified in MB, Cubit will begin to move new volume meshes out to disk automatically.

Saving and Restoring a Cubit Session

There are currently two ways to save/restore a model in CUBIT. A file can be saved with either the Exodus or CUBIT File method. The method of choice is determined by a set command. The CUBIT method is the default.

set save [CUBIT]

CUBIT File Method

- New
- Open
- Close

- Save
- Import
- Export

The CUBIT file is a binary cross-platform compatible file for the storage of a Cubit model that is compact in size and efficient to access. It includes both the geometry and the associated mesh, groups, blocks, sidesets, and nodesets. Mesh and geometry are restored from the Cubit file in exactly the same state as when saved. For example, element faces and edges are persistent, as well as mesh and geometry ids. The Graphical User Interface version of CUBIT also provides a toolbar with direct access to file operations using the CUBIT File method described here.

New

Creates a new blank model with default name, closing the current model. If the user is in the GUI version they will have the option to save the current model before the New command is executed. If the user is in the command-line environment or chooses not to save the current model before starting a new model, the New command basically acts like the reset command.

Open '<filename>'

Opens an existing *.cub file, closing the current model. If the user is in the GUI version they will have the option to save the current model before the Open '<filename>' command is executed.

Close

Closes an existing CUBIT file and assigns it a default name. If the user is in the GUI version they will have the option to save the current model before the Close command is executed. The Close command basically acts like the reset command.

Save

A default file name is assigned when CUBIT is started (in very much the same way the journal files are assigned on startup) in the form cubit01.cub, for example. The current model filename is displayed on the title bar of the CUBIT window. Typing save at any time during your session will save the current model to the assigned *.cub file. The *.cub file includes the *.sat file and the mesh. Groups, blocks, sidesets and nodesets are also saved within the *.cub file. To change the name of the current model, or to save the model's current geometry to a different file, use the save as command. Note that 'save <file.cub>' is NOT a valid command.

save

save as 'filename.cub' [overwrite]

The set file overwrite command can be toggled on and off to allow overwriting when using the save as command. The command is defaulted to not allow overwriting.

set file overwrite [on|OFF]

A backup file is created by default, allowing access to previous states of the model. The backup files are named *.cub.1, *.cub.2... The user can set the total number of backups created per model with the following command (the default number of backups is 99,999):

set save backups <number>

As soon as the number of model backups reaches the maximum, the lowest numbered backup file will be removed upon subsequent backup creation.

To check on the status of a 'set' command, type in the command in question without any options. For example, to check which save method is currently toggled, type:

set save

Import

Appends a *.cub file to an existing model.

import cubit 'filename.cub'

Export

In addition to saving an entire model, one can use the export command to save only a portion of a model. The geometry and associated mesh, groups, blocks, sidesets and nodesets are exported. Only bodies or free surfaces, curves or vertices can be exported to a Cubit file.

```
export cubit 'filename.cub' entity-list
```

Interrupting Running Tasks

Many operations in cubit can be interrupted using **<Control>-C**. Pressing **<Control>-C** will attempt to interrupt the running process as soon as feasible, returning the user back to the command line. This operation will work in both the command line and graphical user interface (GUI) versions of CUBIT. In the GUI version, focus must first be in the graphics window. Click on the graphics window prior to issuing the **<Control>-C** interrupt. Not all operations may be interrupted, and many can only be interrupted at certain stages. Any current tasks are canceled as soon as it is feasible to do so, including playback of journal files. Even if no other running tasks can be canceled, once those tasks are completed playback a journal file is always stopped. Interrupted journal files may be resumed at the next command. See the section titled Controlling Playback of Journal Files for more information on controlling playback of journal files.

Graphical User Interface

- CUBIT Application Window
- Control Panel
- Graphics Window
- Tree View
- Property Editor
- Command Line Workspace
- Journal File Editor
- Toolbars
- Drop-Down Menus

The graphical user interface (GUI) can improve user productivity. It provides an easy way to control CUBIT without learning command syntax. Many geometry commands are faster and easier with the GUI. The underlying GUI components are constructed using a cross-platform development environment. As such, the GUI will behave similarly across all platforms supported by Cubit, yet each GUI will make use of platform specific widgets.

The GUI is built on top of the CUBIT command line. This means that GUI actions are translated to a CUBIT command-line string and journaled. Users familiar with command-line syntax can enter the same text in the GUI command-line window. Journal files can be created and played back in both environments with the same results. Although many things are faster and easier in the GUI, experienced users often use a combination of command line text and GUI button operations.

The discussion of the Graphical User Interface and its features is based on the basic windows contained within the CUBIT GUI Application Window. These are outlined in the subtopics listed above.

A full graphical user interface (GUI) with the standard look and feel consistent with major platforms is available on all supported Cubit platforms. The GUI version can improve productivity, making new users aware of the wide range of CUBIT capabilities, and freeing new and experienced users from having to remember esoteric syntax. The GUI and non-GUI versions create and play back identical journal files, making it easier to switch from one environment to the other.

CUBIT Application Window

The default CUBIT Application Window is shown in the following image.

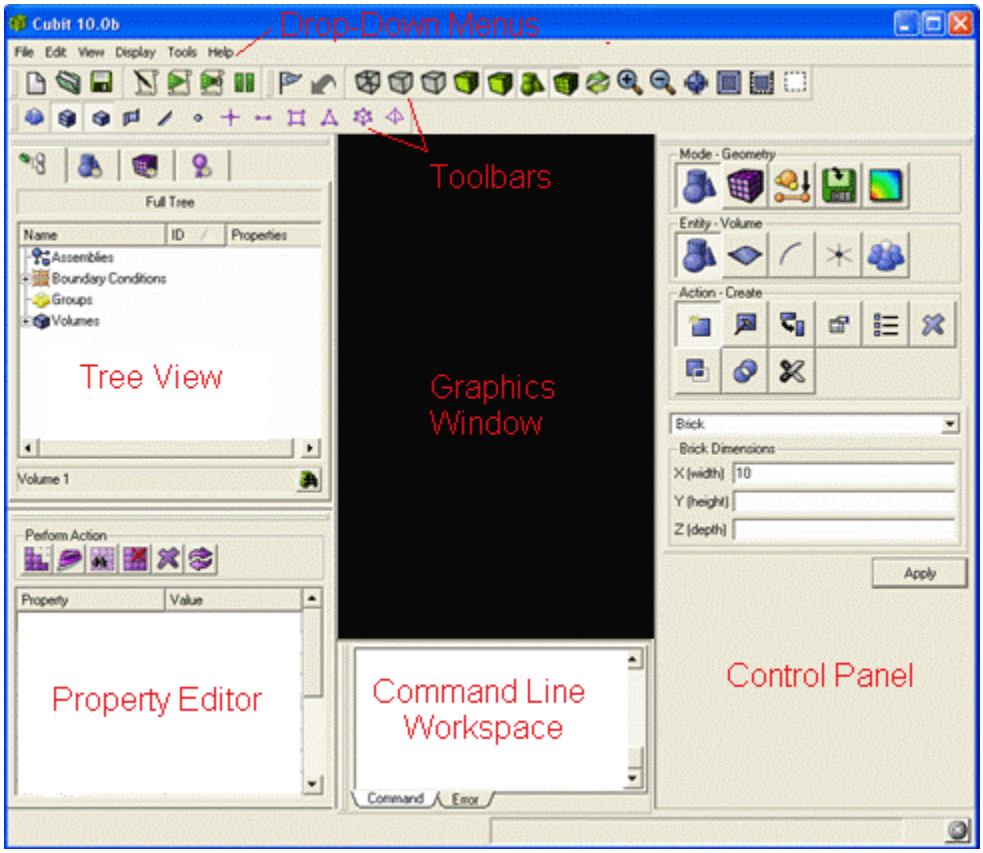


Figure 1. The CUBIT Application Window

Graphics Window- The current model will be displayed here. Graphical picking and view transformations are done here.

Tree View - Geometry tree hierarchy view, geometry analysis and repair tool, meshing tool, and meshing quality tool

Property Editor - The Property Editor lists attributes of the current entity selection. Some of these properties can be edited from the window.

Control Panel - Most Cubit commands are available through the command panels. The panels are arranged topologically, by mode.

Command Line Workspace - The command line workspace contains both the cubit command and error windows. The command window is used to enter cubit commands and view the output. The error window is used to view cubit errors.

Pull-Down Menus - Standard file operations, Cubit setup and defaults, display modes, and other functionality is available in the pull-down menus.

Toolbars - The most commonly used features are available by clicking toolbar icons.

Customizing the Application Window

All windows in the CUBIT Application can be *Floated* or *Docked*. In the default configuration, all windows are docked. When a window is *docked* the user can click on the area indicated below.

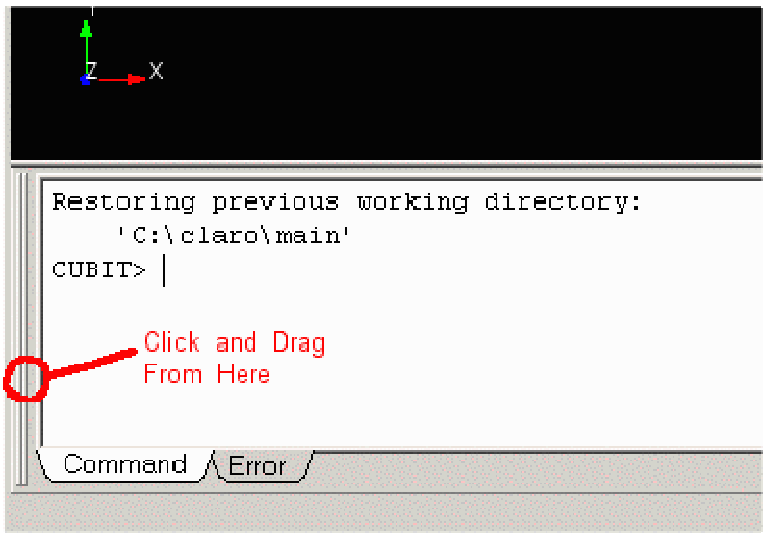


Figure 2. A docked window. Click and drag to float.

By dragging with the left mouse button held down, the window will be un-docked from the Application Window. Dragging the window to another location on the Application Window and releasing the mouse button will cause it to dock again in a new location. The bounding box of the window will automatically change to fit the dimensions of the window as it is dragged. Releasing the mouse button while the window is not near an edge will cause the window to Float. To stop the window from automatically docking, hold the CONTROL key down while dragging.

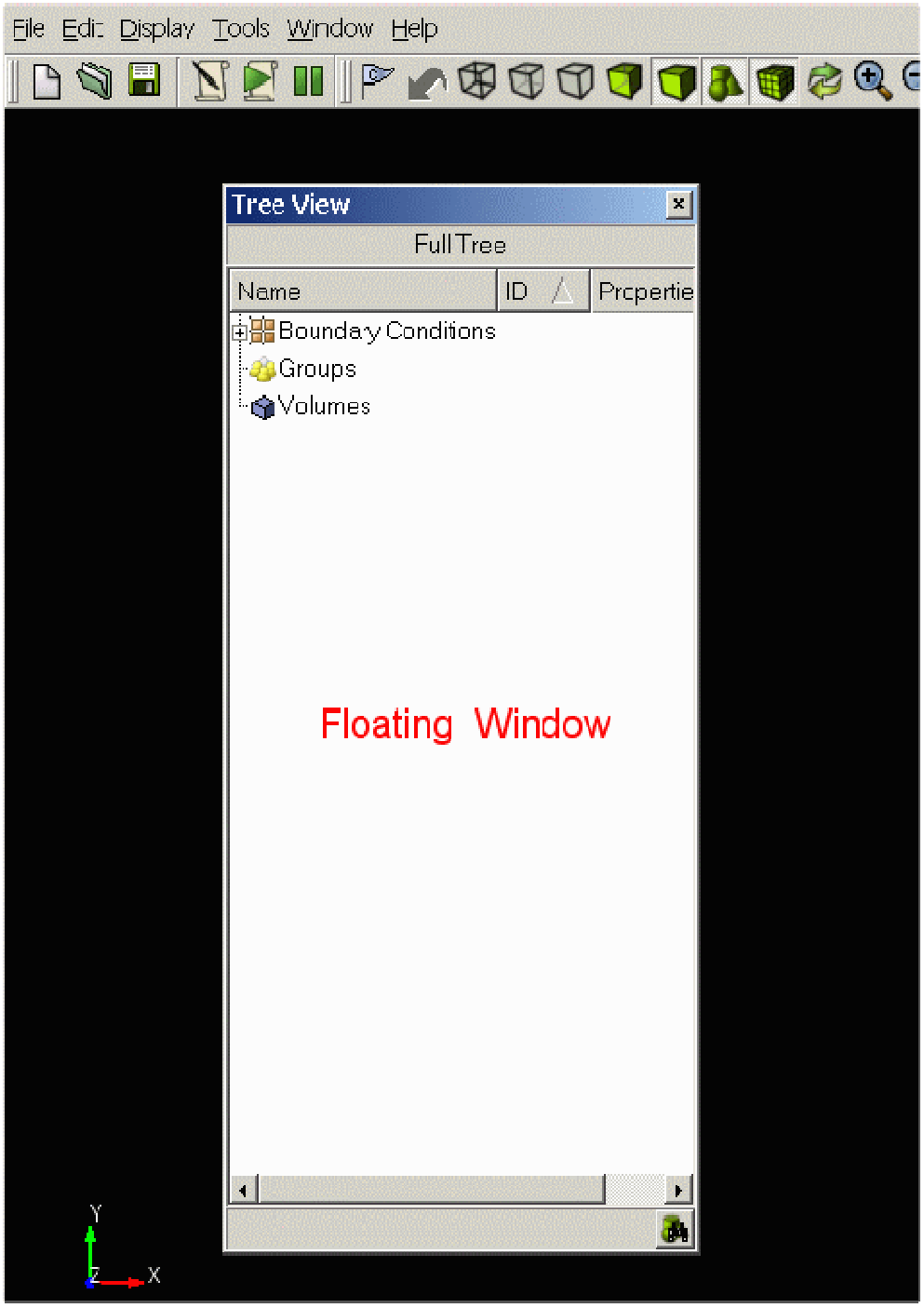


Figure 3. A Floating Window

When a window is *floating*, as shown in Figure 3, it is possible to dock it by clicking the title bar of the window and dragging it to its new docked location.

Note: Double clicking on the title bar of an floating window will cause the window to *redock* in its last docked position.

Control Panel

The Control Panel provides a graphical means of accessing almost all of the CUBIT functionality. The main CUBIT Control Panel is divided into six modes. Each of these modes pertains to a major component of the CUBIT application. This documentation is designed to provide information on interaction with the GUI and to provide links to documentation about every feature that can be accessed from the GUI. Follow the links as you would on the GUI to access help about each command. To view information about each of the tools in the Control Panel select the icon on the image below.

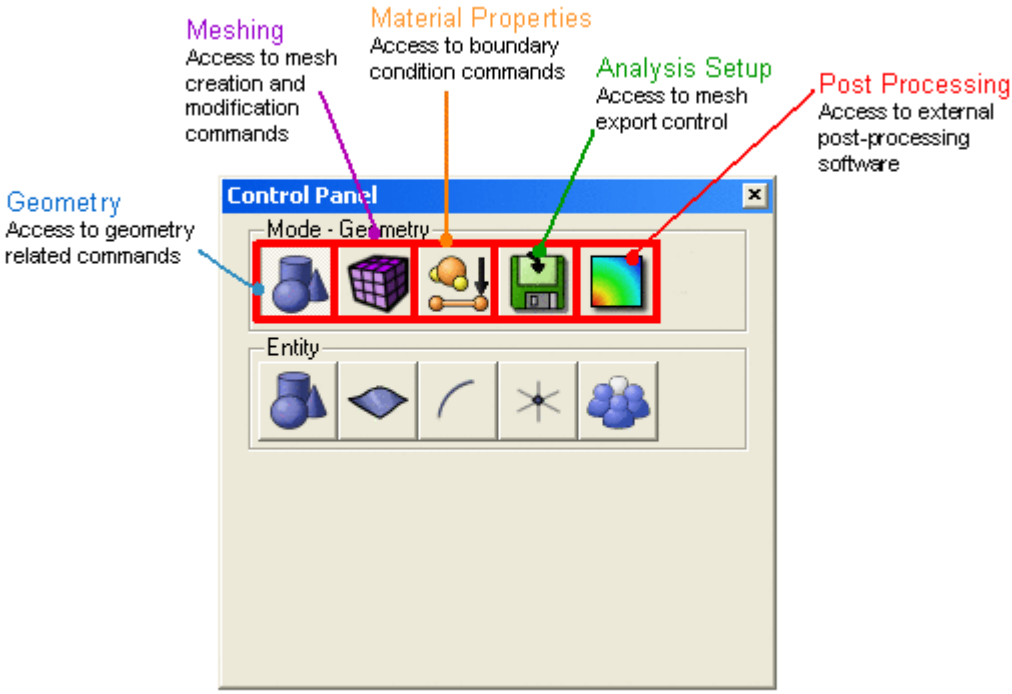


Figure 1. The CUBIT Control Panel

A brief description of the functionality of the Control Panel window, along with a graphical overview to the GUI widgets follows.

Control Panel Functionality

Graphical Overview to GUI

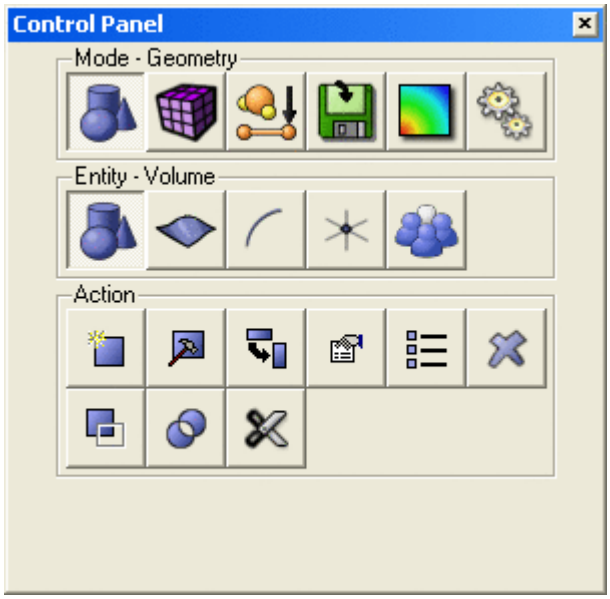
Control Panel Functionality

The Control Panel is arranged first by mode on the top row of buttons. Modes are arranged by task. All of the geometry related tasks, for instance, can be found under the Geometry mode. When a mode is selected, a second row of buttons becomes available. The second row of buttons shown depend on the selected mode. For example, if Geometry, Meshing, or Materials and Properties is selected, the second button row will show Entities (as in the example shown above.) Entities are those specific to the mode.

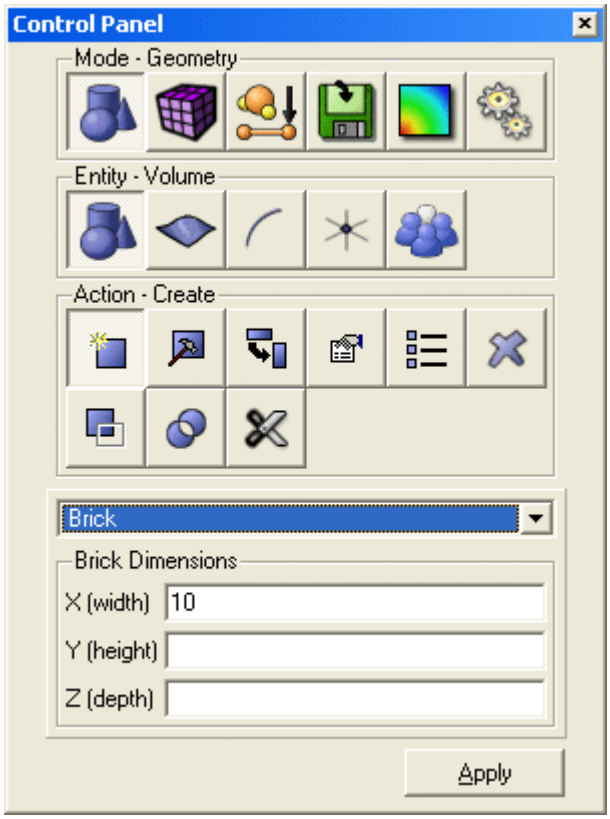
- Geometry entities include Volumes, Surfaces, Curves, Vertices, and Groups.
- Meshing entities include Volumes, Surfaces, Curves, Vertices, Groups, Hexes, Tets, Quads, Tris, Bars, and Nodes.
- Materials and Properties entities include Element Blocks, Sidesets, and Nodesets.

The second row of buttons for Analysis Setup, Post Processing, and Power Tools are not arranged by entity. Rather, the buttons show specific capabilities.

The third row of buttons are Actions, such as Create, Delete, Modify, and so forth. The following shows an example of Geometry/Volume actions.



Selecting an Action will display a command panel. The Geometry/Volume/Create command panel is shown below.



All command panels are constructed similarly. Each abstracts a set of Cubit commands. Options are selected using checkboxes, radio buttons, combo boxes, edit fields, and other standard GUI widgets. Each command panel includes an Apply button. Pressing the Apply button will generate a command to Cubit. Nothing happens until and unless the Apply button is pressed.

> **Note: The edit fields are free form, meaning, the user may enter any valid string into the fields. Any string that is valid for the command line is valid for the command panel edit fields.**

Where possible, default values are placed into edit fields. At any time, with the cursor placed over a blank portion of the command panel, the user may right-click to select Reset Data which will clear all fields and replace default values.

ID Input Entry Methods

The *ID Input Fields* provide a location where Geometric IDs, required for the current command, can be entered. IDs can be entered in several ways:

Simple Keyboard entry

ID numbers can be entered directly in the field. Each ID must be separated with a space. Select the field first before typing.

Graphical selection

IDs can be entered automatically by selecting entities directly in the Graphics Window. The current entity available for selection is based on the current entity selection mode. In some cases, not all entities of the current entity selection mode will be available for picking. The program may automatically filter the applicable entities based on the context of the current command

Geometry Tree selection

IDs may be entered by selecting the corresponding geometric entity from the geometry tree. To select multiple entities use the **<ctrl>** key.

Ranges

A range of IDs may be typed into the field. For example:

> **1 to 5**

will automatically enter all IDs from 1 to 5 inclusive in the field. Keywords such as **all** and **except** can also be used. Any range that can be entered directly on a CUBIT command line can also be used in the ID input field. See Command Line Entity Specification for a description of the syntax.

As Part of Other Entities

Syntax can be entered in the *ID Input field* that will specify an entity based upon its topological relationship to other entities For example, if a **Vertex** Selection Type Button was highlighted, entering

> **in surf 1**

will automatically enter all vertices in surface 1 into the *Input Field*. CUBIT has a rich set of syntax rules for specifying entities based upon topology relationships. See Command Line Entity Specification for a description.

In Groups

Entities that are part of groups may be specified in the ID Input Field. For example, if the Vertex Selection Type Button is highlighted, entering:

> **in picked**

will automatically enter all vertices in the picked group into the active *ID Input Field*.

Dragged and Dropped

Entities can be selected in other windows and dragged and dropped into the *ID Input Field*. This is frequently done by selecting text within the command window and dragging and dropping the text into the *ID Input Field*. For example, selecting the text "**Curve 1**" in the command window and dropping it into the *ID Input field* will enter a **1** in the field. The current selection type must match the entity being dropped for this method to be effective.

Right-Click Context Menu for ID Input Fields

When the right mouse button is selected while in an *ID Input Field*, the following menu options will appear:

- **Done Selecting** - Enters current selection and removes cursor from selection window
- **Select Other** - Displays selection dialog
- **Insert Selection** - Echoes current picked entity to the selection window
- **Reject Selection** - Clears current selection
- **Select All** - Selects all available entities and puts "select all" in input window
- **Clear Selections** - Clears the entire selection window
- **Highlight -** Highlight the current selection
- **Zoom To** - Zooms to current entity in the selection field within the graphics window
- **Draw** - Draws the entities listed in the input field within the graphics window
- **Isolate** - Turns visibility off for all entities other than the selected entities. Similar to draw command, but entities remain hidden with a graphics refresh. Select **All Visible** in the graphics window to turn visibility back on.
- **Visibility Off** - Removes the current entity from the input window and hides it on the graphics screen
- **Measure** - Displays a sub menu of choices. Replaces the selected entity(s) in the ID Input Field with the item selected from the menu.
- **Mesh** - Mesh the listed entities using either an assigned scheme or a default scheme where none is assigned
- **Delete Mesh** - Deletes mesh on all entities listed in the input window
- **Reset Entity** - rehighlights the entities listed in the input field within the graphics window
- **List Info** - Displays a sub menu of choices including basic, geometry, and mesh. Selecting the basic option will list schemes, visibility, and interval assignments. The geometry option will add information about the geometry and geometry engine. The mesh option will list information about mesh entities.
- **Delete** - Deletes the current geometric object in the input window.

Value Fields

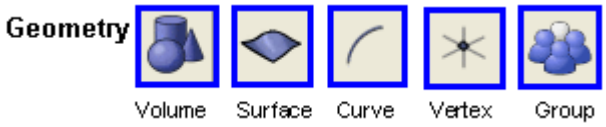
Integer and real values pertinent to the command are entered in this window. Input placed in parenthesis { } will be evaluated when the command is executed. For example:

{10*0.02}

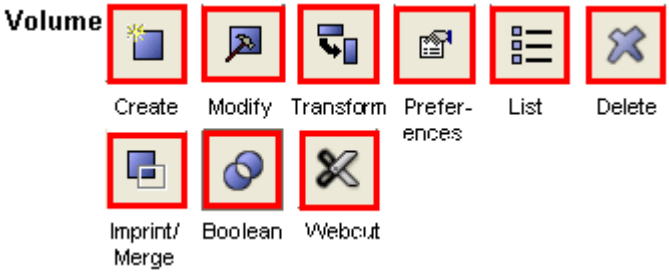
is valid input. Additionally, any APREPRO syntax is valid in the *Value Field*, including mathematical functions and boolean operations. See the section, APREPRO for a description of syntax.

Control Panel Overview

This page gives a graphical overview to all of the Control Panel functionality. Buttons are grouped by Modes, Entities, and Actions as they are on the actual Control Panel. Clicking on a selection will take you to the next submenu as if you were in the GUI until you reach applicable documentation describing each of the specific commands. To help visual this page, first level modes are highlighted in green, second level entities in blue, and third level actions in red. If there are additional menus within the action level, they are also listed.



The Geometry Mode on the Graphical User Interface is divided into groups of actions that apply to each geometric entity. These entities include: Volume, Surface, Curve, Vertex, and Group. All of the commands that can be accessed from the Geometry Mode are listed below.



- **Create**: Brick, Cylinder, Sphere, Prism, Pyramid, Cone, Torus, Copy, Loft, Bounding Surfaces, Sweep
- **Modify**: Regularize, Separate, Split Periodic, Section, Heal
- **Transform**: Align, Move, Scale, Rotate, Reflect
- **Preferences**: Label, Visibility, Color
- **List**
- **Delete**
- **Imprint/Merge**: Imprint/Merge, Unmerge
- **Booleans**: Unite, Subtract, Intersect.
- **Webcut**: Chop, Plane, Plane Vertex, Plane Surface, Plane Normal to Curve (near vertex), Plane Normal to Curve (vertex), Tool, Sheet, Sheet Extended from Surface, Sweep Curve, Sweep Surface, Cylinder Radius, Loop

- **Create**: Vertex List, Bounding Curves, Copy, Extended Surface, Planar Surface, Net Surface, Offset, Skin Curve, Sweep, Midplane
- **Modify**: Regularize, Split, Composite, Partition, Tweak, Remove.
- **Transform**: Move, Reflect, Rotate, Scale.
- **Preferences**: Label, Color
- **List**
- **Delete**
- **Merge**

- **Create**: Straight, Spline, From Curves, Offset, Arc, Combine, Copy, Onto Curve, Project, Arc or Circle
- **Modify**: Regularize, Composite, Partition, Split, Trim, Blend/Chamfer.
- **Transform**: Move, Reflect, Rotate, Scale.

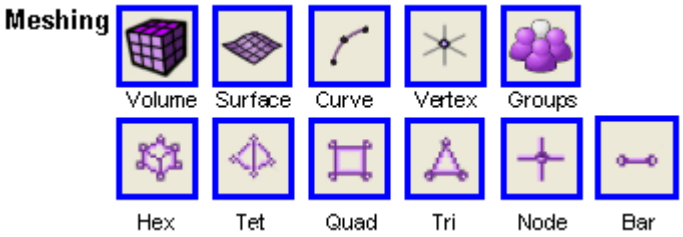
- **Preferences**: Label, Color
- **List**
- **Delete**
- **Merge**

- **Create**: Coordinates, Arc Center, On Curve, At Intersection, Copy(create on vertex), Picking.
- **Modify**: Regularize, Move, Collapse Angle
- **Merge**
- **Preferences**: Label, Visibility, Color

  List

- **Delete**

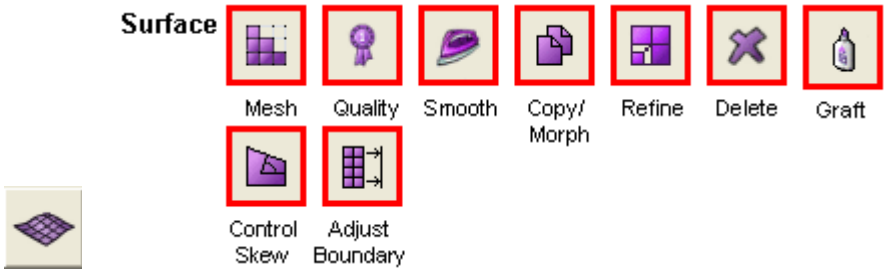
- **Manage Groups**: Create/Add, List, Delete/Remove
- **Webcut**: Plane, Plane Vertex, Plane Surface, Plane Normal to Curve (near vertex) , Plane Normal to Curve (vertex), Tool, Sheet, Sheet Extended from Surface, Cylinder Radius



The Meshing Control Panel contains all of the mesh creation and modification commands. It is divided by geometric and mesh entities.

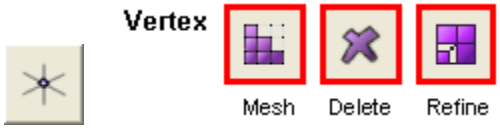
- **Mesh:**
  - **Interval Functions**: Constant Size, Interval, Auto, Geometry-adaptive
  - **Schemes**: Map, Submap, Sweep, Tetmesh, TetPrimitive, Sphere, Polyhedron, Automatically Calculate.

- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Robinson, Traditional, Aspect Ratio Bet, Aspect Ratio Gam, Aspect Ratio, Condition No., Diagonal Ratio, Dimension, Distortion, Element Volume, Jacobian, Relative Size, Scaled Jacobian, Shape and Size, Shear and Size, Shear, Skew, Stretch, Taper, Warpage
  - **Options**: Filter, Draw, List, Scope
- **Smooth**: Laplacian, Conditon No, Equipotential, Untangle, Optimize Jacobian, Mean Ratio
- **Copy/Morph**
- **Refine**
- **Delete**
- **Graft/Meshcut**: Meshcut, Graft



- **Mesh**:
  - **Interval Functions**: Constant Size, Interval, Auto, Geometry-adaptive
  - **Schemes**: Map, Pave, TriMesh, Submap, TriPrimitive, Circle, Hole, Mirror, Pentagon, TriDelaunay, Automatically Calculate.
- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Robinson, Traditional, Aspect Ratio, Condition No., Distortion, Element Area, Jacobian, Max Angle, Min Angle, Relative Size, Scaled Jacobian, Shape and Size, Shear and Size, Shear, Skew, Stretch, Taper, Warpage
  - **Options**: Filter, Draw, List, Scope
- **Smooth**: Centroid Area, Laplacian, Conditon No, Untangle, Winslow, Mean Ratio
- **Copy/Morph**
- **Refine**
- **Delete**
- **Graft**
- **Control Skew**: Control Skew, Delete Skew Control
- **Adjust Boundary**

- **Mesh**:
  - **Interval Functions**: Constant Size, Interval, Auto
  - **Mesh Functions**: Equal, Bias, Curvature, PinPoint, Stretch, Featuresize
- **Smooth**: Laplacian
- **Copy/Morph**
- **Refine**
- **Delete**



- **Mesh**
- **Delete**
- **Refine**

- **Mesh:**
  - **Interval Functions**: Depends on entities in group
  - **Schemes**: Depends on entities in group
- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Robinson, Traditional, Aspect Ratio Bet, Aspect Ratio Gam, Aspect Ratio, Condition No., Diagonal Ratio, Dimension, Element Area, Element Volume, Jacobian, Max Angle, Min Angle, Relative Size, Scaled Jacobian, Shape and Size, Shear and Size, Shear, Skew, Stretch, Taper, Warpage
  - **Options**: Filter, Draw, List, Scope
- **Smooth**: Depends on entities in group

- **Smooth**
- **Quality**:
  - Metrics: Shape, Algebraic, Allmetrics, Robinson, Traditional, Aspect Ratio, Condition No., Diagonal Ratio, Dimension, Distortion, Element Volume, Jacobian, Relative Size, Scaled Jacobian, Shape and Size, Shear and Size, Shear, Skew, Stretch, Taper, Warpage
  - **Options**: Filter, Draw, List, Scope
- **Refine**

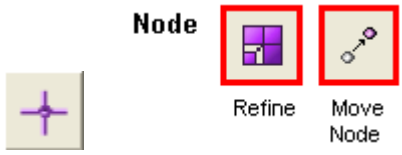
- **Smooth**
- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Traditional, Aspecct Ratio Bet, Aspect Ratio Gam, Condition No., Distortion, Element Volume, Jacobian, Relative Size, Scaled Jacobian, Shape and Size
  - **Options**: Filter, Draw, List, Scope
- **Refine**

- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Robinson, Traditional, Aspect Ratio, Condition No., Distortion, Element Area, Jacobian, Max Angle, Min Angle, Relative Size, Scaled Jacobian, Shape and Size, Shear and Size, Shear, Skew, Stretch, Taper, Warpage
  - **Options**: Filter, Draw, List, Scope

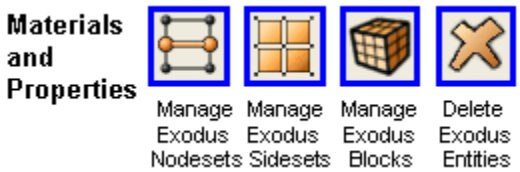
- **Refine**

- **Quality**:
  - **Metrics**: Shape, Algebraic, Allmetrics, Traditional, Condition No.,Distortion Element Area, Jacobian, Max Angle, Min Angle, Relative Size, Shape and Size
  - **Options**: Filter, Draw, List, Scope
- **Refine**

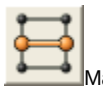
- **Refine**



- **Refine**
- **Move Node**



The Materials and Properties command panel contains buttons for creating, editing, and deleting exodus nodesets, sidesets, and blocks.

 Manage Exodus Nodeset

- Add, Color, Draw

Manage Exodus Sidesets

- Add, Patch, Color, Draw

Manage Exodus Blocks

- Add, Color, Element Type, Attribute, Attribute by Index

Delete Exodus Entities

- Nodesets and Sidesets, Blocks

The Analysis Setup Menu controls export mesh commands and quality filters.

Export Mesh: Genesis, Abaqus, Ansys, I-DEAS Universal, NASTRAN BDF, Patran

Filter Failed Elements

Post Processing

The Post-Processing button is used to link in external post-processing components for visualization of results. You must first set the path to the post-processing software in the Tools-Option dialog box.

Graphics Window

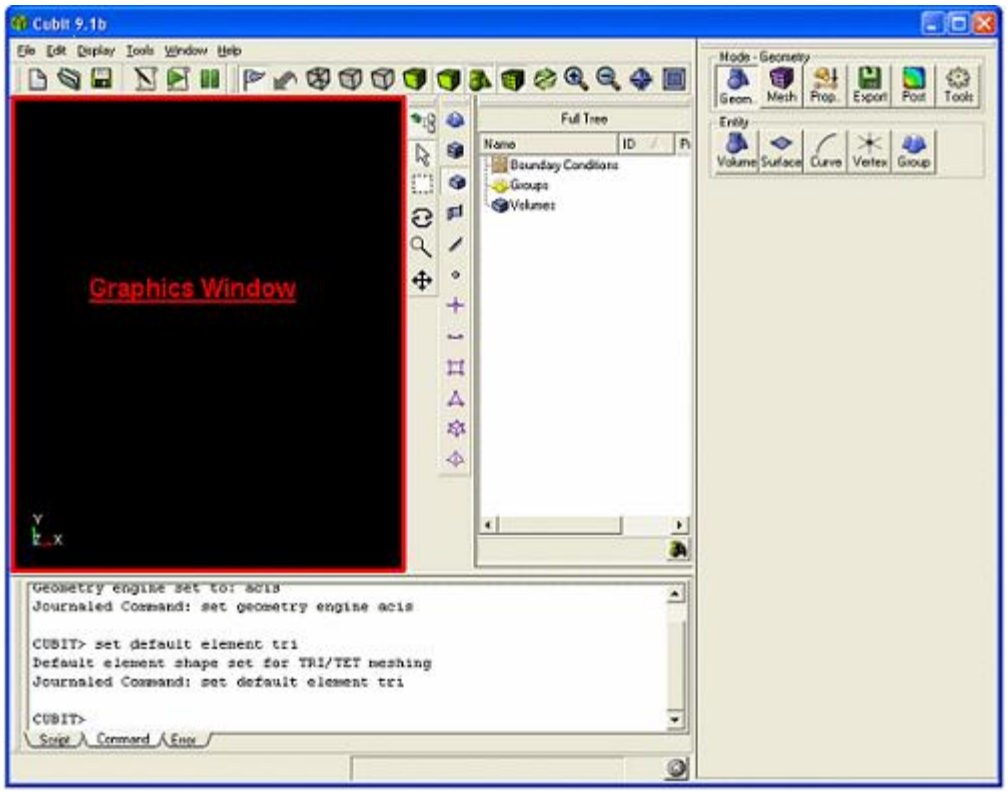


Figure 1. Graphics Window

The graphics window is used to view and select entities. Select one of the options below:

- View Navigation
- Selecting Entities
- Key Press Commands
- Changing the Display Mode
- Right Click Commands

View Navigation in the GUI

The following table summarizes the mouse-based view navigation operations in the GUI. (See Mouse-Based Navigation for the command line version). There are two different default paradigms: Cubit command line and Cubit GUI. The user is allowed to customize the mouse settings as desired. Mouse settings are modified by accessing the **Tools** pull-down menu, then select **Options**. The Mouse Settings dialog is shown below.

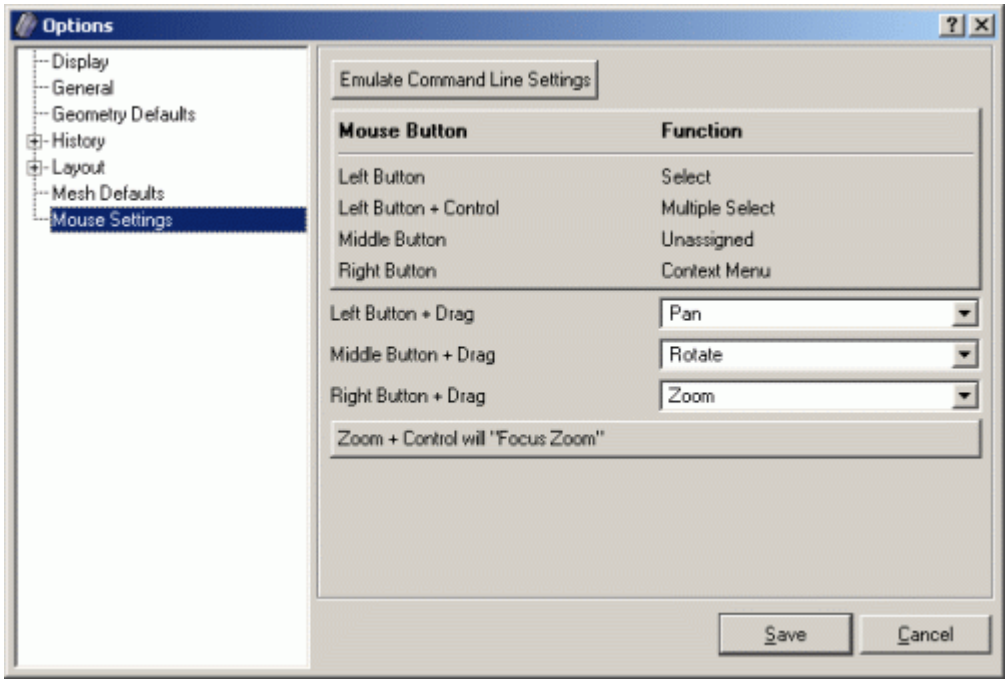


Figure 1. Mouse Settings Dialog

Rotations

Where the cursor is in the graphics window will dictate how the view will be rotated. If the cursor is outside of an imaginary circle, shown in Figure 1, the view will be rotated in 2d, around an axis normal to the screen. If it is inside the circle, as in Figure 2, the rotations will be in 3d, about the current view spin center. The spin center can be changed to any x-y-z location. The most common way is by zooming to an entity, which changes the spin center to the centroid of that entity. The "view at" command will change the spin center without zooming:

View at vertex 3

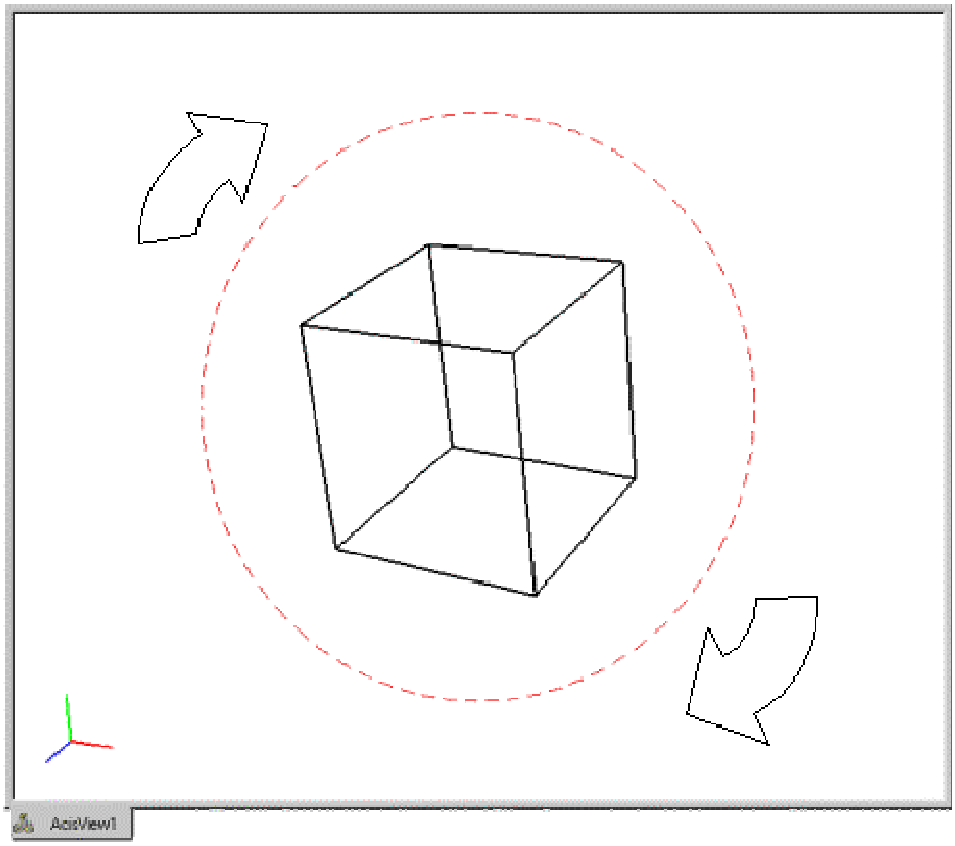


Figure 2. With the mouse pointer outside the circle the view is rotated about an axis normal to the screen

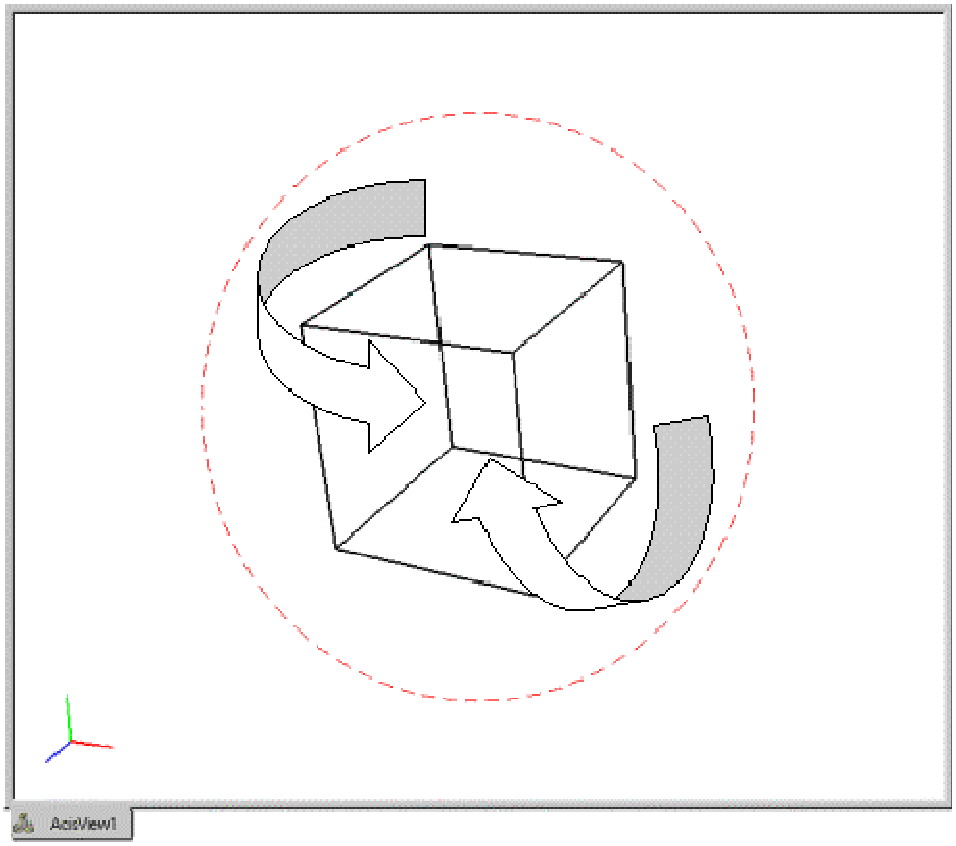


Figure 3. With the mouse pointer inside the circle the view is rotated about the current spin center

Zooming

To zoom, press the appropriate buttons or keys and move the cursor vertically, as shown in Figure 3.

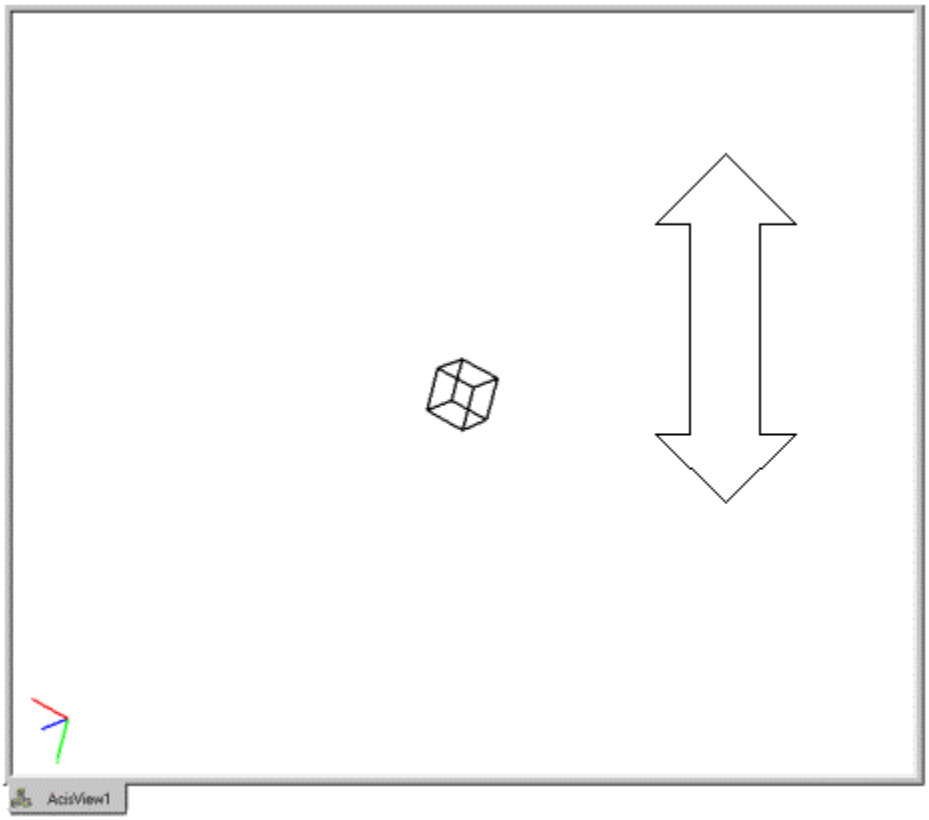


Figure 4. Move the mouse pointer vertically to zoom in and out

Panning

To pan, press the appropriate buttons or keys and move the cursor horizontally or vertically, as shown in Figure 4.

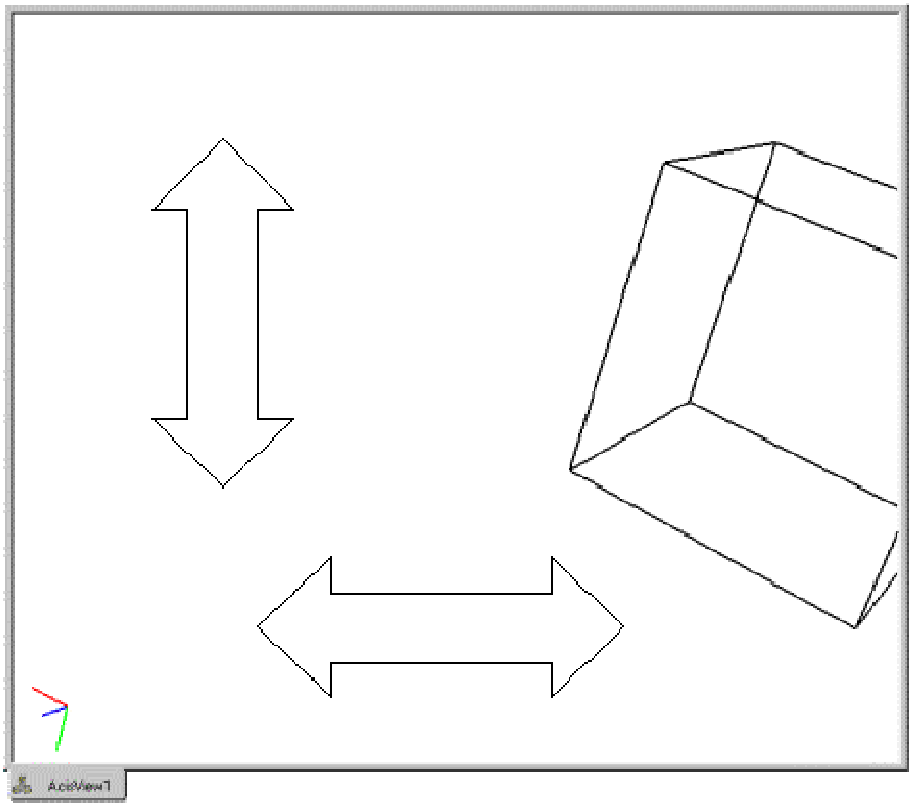


Figure 5. Move the mouse pointer horizontally or vertically to pan the view

Selecting Entities in the GUI

Geometry and Mesh Entities can be selected with the left mouse button directly on the graphics window. Before selecting any entity, however, the correct selection mode must be chosen. This dictates which entity types will be available for selection in the graphics window. The *Select* Toolbar, which is located on the right of the graphics window by default, is used to change the entity selection modes.

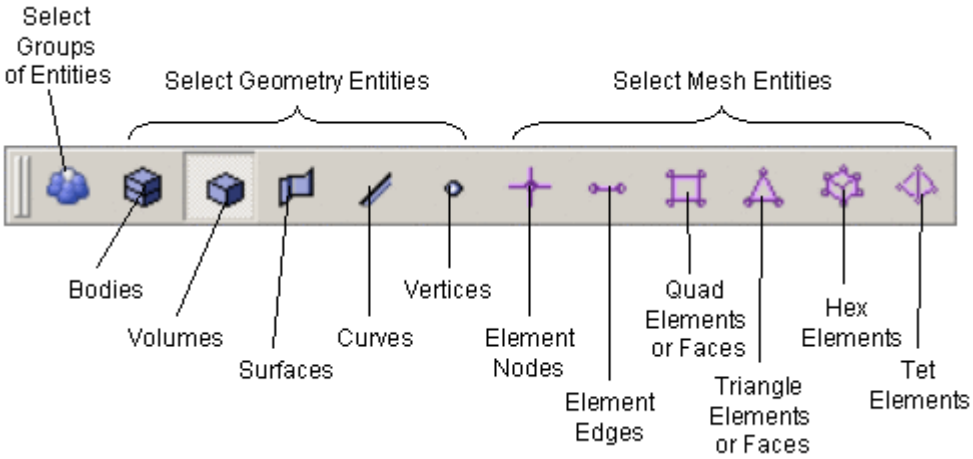


Figure 1. The Select Toolbar

Figure 1, shows the Select toolbar. Selecting one of the entity selection modes will only permit selection of that particular entity type within the graphics window. These selections will not override a Pick Widget in the command panel.

When the mouse cursor is over an entity type that has been selected from the Pick toolbar, that entity will become highlighted. This is called **pre-selection** and is used as a graphical guide to show which entity will be picked when the mouse button is clicked.

If both volumes and surfaces entities are picked on the Select toolbar, a single click will select the surface while a double click will select the volume.

More detailed information on selecting and specifying entities can be found in Entity Selection and Filtering .

Key Press Commands for the GUI

Several commands have a key press shortcut. These commands will be executed with respect to the currently selected entities; see the following table:

Shortcut Key	Command
l	List information about the current entity to the output window.
i	Toggle the visibility of the selected entity (make invisible or visible).
e	Echo entity id to command line.
Tab	Select the next entity.
Shift Tab	Select the previous entity.
0	Set pick type to vertices.
1	Set pick type to curves.
2	Set pick type to surfaces.
3	Set pick type to volumes.
4	Set pick type to groups.
Shift **0**	Set pick type to mesh nodes.
Shift **1**	Set pick type to mesh edges.
Shift **2**	Set pick type to mesh faces.
Shift **3**	Set pick type to mesh hexes.

F5	Refresh graphics window

Changing the Display Mode in the GUI

The display mode can be changed by using icons in the Display toolbar. By default, this toolbar is located above the graphics window.

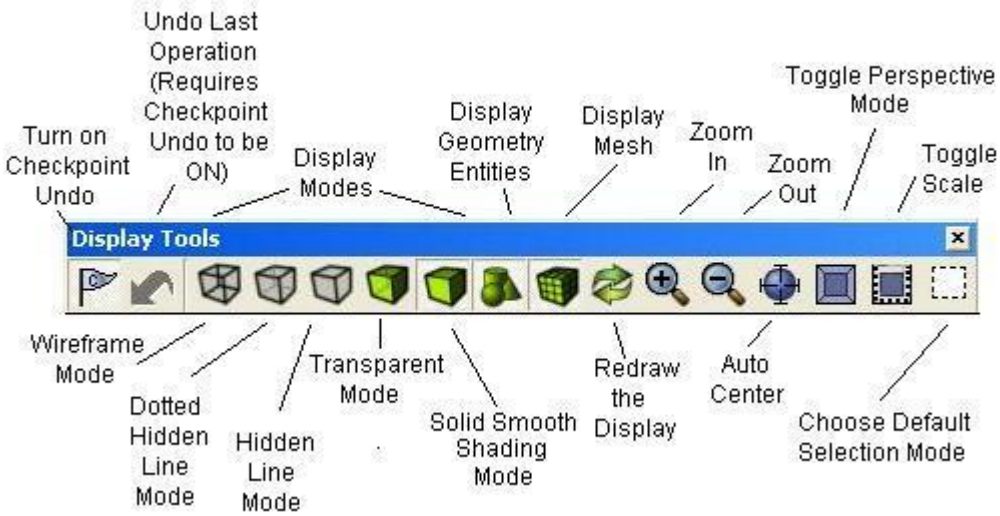


Figure 1. The Display Toolbar

The function of each of the buttons on the toolbar, as with all buttons in CUBIT, can be displayed by putting the mouse cursor over the button.

Icons 3 through 9 in the Display toolbar change the Display mode. The following is a brief description of each of the buttons in the Display toolbar. For a more detailed discussion of the graphics display options, see Graphics Modes.

Display Mesh - Toggle Display of the model's mesh.

Display Geometry - Toggle Display of the model's geometry.

Shaded - Surfaces are filled and shaded. Works best with OpenGL-supported video cards.

Transparent - Surfaces are drawn semi-transparent.

Hidden Line - Surfaces not visible are not drawn.

True Hidden Line - Surfaces not visible are drawn with dashed lines.

Wire Frame - Surfaces are completely transparent. This is the fastest graphics mode.

Right Click Commands for the GUI Graphics Window

Clicking the Right mouse button in the graphics window will bring up a menu. One of two menus will appear, depending on whether an entity is currently selected.

With Entity Selected

- Zoom To - Zoom to the selected entity
- Draw - Draw the selected entity
- Isolate - Turn all but the selected entities invisible
- Visibility Off - Turn selected entities invisible
- Measure - Measure distance between two selected entities, or measure the length of a selected curve.
- Mesh - Mesh the selected entities
- Delete Mesh - Delete the mesh on selected entities
- Reset Entity - Reset selected entities
- List Info - Show the menu of additional list commands
- Delete - Delete selected entities

Without Entity Selected

- Reset Zoom
- Refresh
- All Visible
- Background
- Display Options

Tree View

While the Control Panel is the best place to look for task-oriented procedures, some operations are best performed from an entity-oriented approach. The Tree View and Property Editor are designed as an entity-oriented tools. The Tree View contains a complete hierarchical representation of all entities, as well as tools designed to analyze, repair and improve geometry and mesh elements. The Tree View is divided into four tabs. These are:

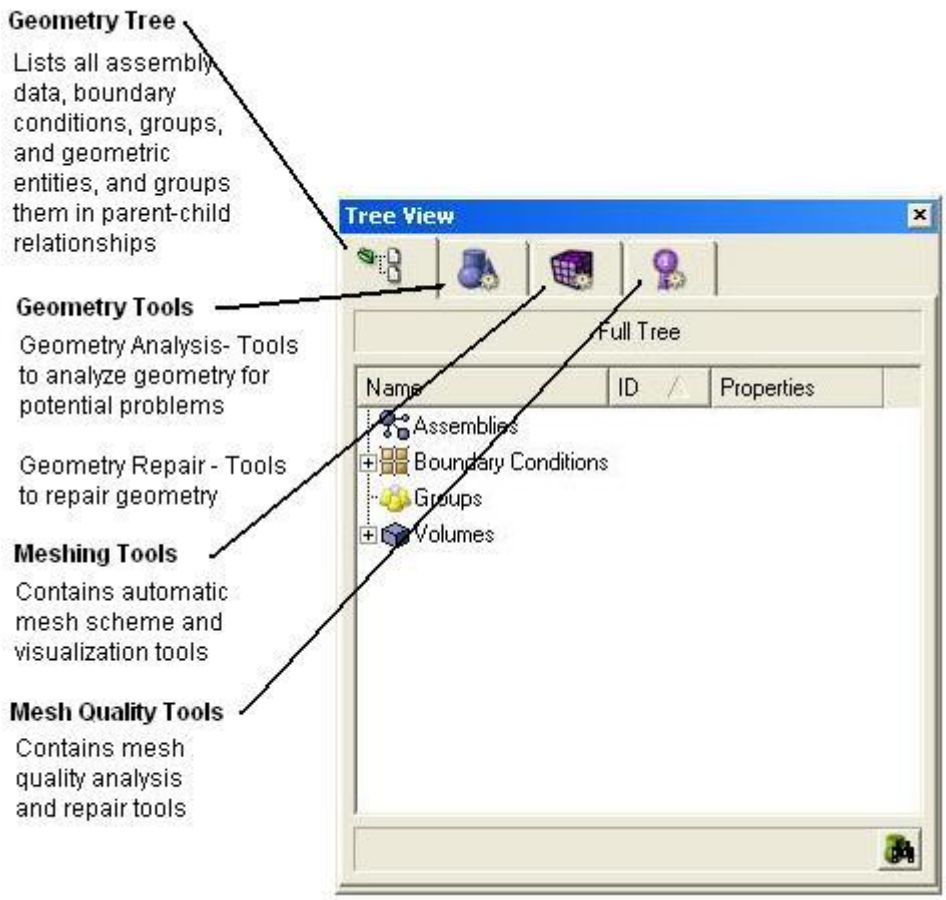


Figure 1. Tree View Window

Geometry Tree

Geometry Analysis and Repair Tools

Meshing Tools

Mesh Quality Tools

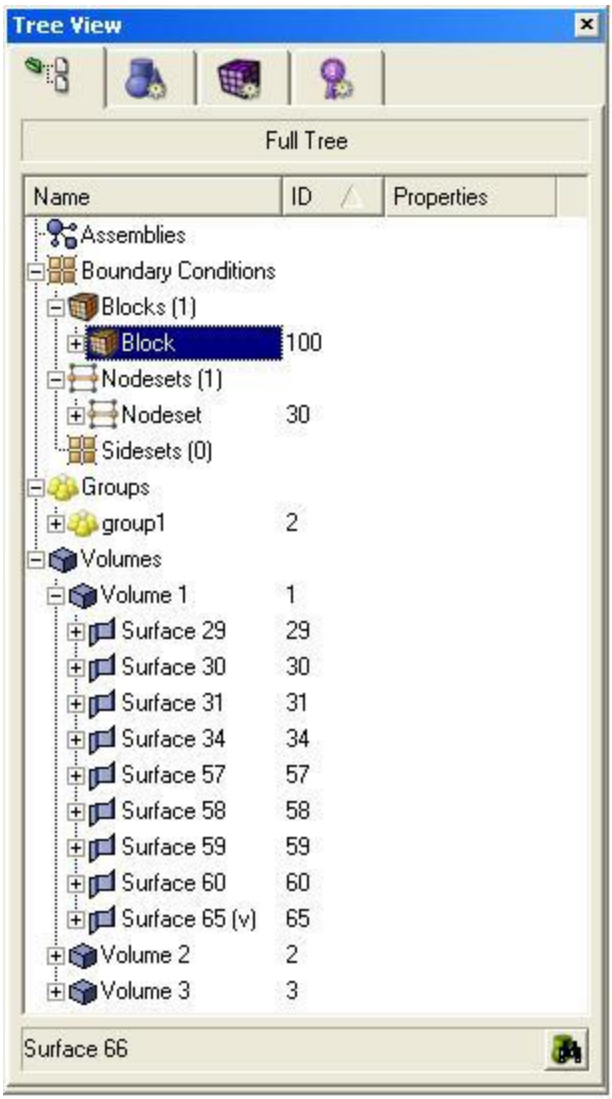
Geometry Tree

The geometry tree provides a complete graphical hierarchical representation of the parent child relationship of all geometric entities. The tree is populated as the model is constructed by Cubit. In addition to showing a hierarchy of geometric entities, the tree also shows Assembly Data, active Groups, and active Boundary Condition entities.

The tree works directly with the graphics window and picking. Selecting an entity in the tree will select the same entity in the graphics window. Selecting an entity in the graphics window will highlight the tree entry if that entry is currently visible.

If the tree entry is not visible the user may press the Find button located at the bottom of the tree. The first occurrence of the selected entity will be shown on the tree.

Virtual entities have a small (v) after the name to indicate that they are virtual entities.



Drag and Drop

The Tree View window supports drag and drop of geometric entities into existing boundary condition sets. To create blocks, sidesets, or nodesets, see the Materials and Properties menu on the main control panel, or right-click on the Blocks (parent) label and select Create New Block. Geometric entities can be added to blocks, nodesets, or sidesets by dragging and dropping inside the tree view window. Groups can also be dragged into boundary condition sets.

Right-Click Menu Functions

The geometry tree's context menu is sensitive to the type of item and the number of items selected. Functions that apply to the item type and number of selected items are available from the context menu.

- **Zoom To** - Available for all geometric entities
- **Draw** - Draw this entity by itself.
- **Isolate** - Similar to Draw command, but the display will not be refreshed with a graphics reset. To redisplay the model, select All Visible from the graphics window right-click menu.
- **Transparency On/Off** - Toggles transparency mode
- **Visibility On/Off** - Toggles visibility
- **Rename** - Allows you to rename entities from the tree. Clicking on a highlighted entity in the tree will do the same thing.

- **Mesh** - Mesh selected entity at current settings.
- **Delete Mesh** - Available for meshed entities/li>
- **Reset Entity** - Deletes mesh, and returns all settings to default values.
- **Delete** - Available when Volumes and Groups are selected.
- **Measure** - Available when two entities are selected or 1 curve is selected
- **Refresh Full Tree** - Used to return to main tree
- **Collapse Tree** - Available when entities are selected.
- **View Descendants/Ancestors** - Show this entities individual hierarchy. Use the Refresh Full Tree option to return to main tree view.
- **View Neighbors** View adjacent entities. Use the Refresh Full Tree option to return to the main tree view.
- **Create New Volume** - Available when the user right-clicks over the Volumes (parent) label.
- **Import Geometry** - Available when the user right-clicks over the Volumes (parent) label. Opens import dialog.
- **Create New Group** - Available when the user right-clicks over the Groups (parent) label.
- **Clean Out Group** - Available when groups are selected. Removes all entities from group.
- **Create New Block/Sideset/Nodeset** - Available when the user right-clicks over the respective Boundary Conditions (parent) label.
- **Remove from Block/Sideset/Nodeset** - Removes selected entity from the specified block, sideset or nodeset.
- **Update BC Data** - available when the user right-clicks over the Boundary Conditions (parent) label. NOTE: Boundary Condition data is not currently automatically updated in the tree. To ensure the user is viewing the most current BC data, select Update BC Data.
- **Other Views** - Additional tree views including non-hierarchical views, and sweep surface lists.
- **List Info** - List information about selected entity in the output window.

Geometry Power Tools

The geometry power tools are located on the Tree View window under the blue geometry tab. In many cases, a model will fail to mesh because of problems with the geometry. Since the range of geometry problems is so wide, and because these problems can be hard to diagnose, the Geometry Power Tool has several built-in tools designed to analyze and repair these problems. The Geometry Repair Tool analyzes geometry for small angles, overlap, small features, bad geometry definition, blend surfaces, close loops, or mergeable entities that may affect meshing capability. It also contains a powerful toolkit of geometry modification methods to fix these problems. All of the common geometry clean-up tools are now in one place on the GUI menu. In addition, there is a window that lists results from geometry analysis in a tree format, making it easier to find, diagnose, and solve geometry problems.

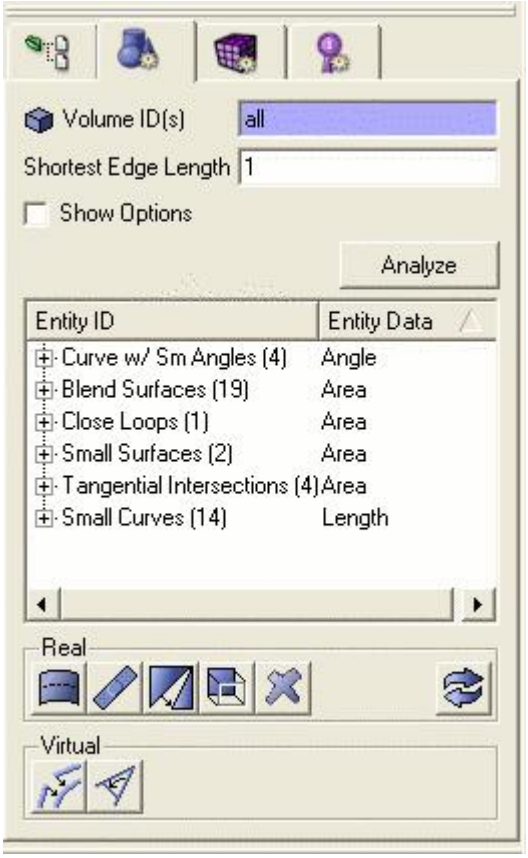


Figure 1. Geometry Power Tools

Geometry Analysis Tools

The geometry power tools contain an array of tests that can be run on geometry to diagnose potential problems for mesh generation. To display a list of tests, click on the **Show Options** check box. By default all tests are selected and run on geometry. Some tests may not apply to specific geometry, or may only need to be run once per geometry (i.e. bad geometry definition test). Clicking on the box by each test will deselect it.

The geometry analysis inputs and tests are summarized below:

Shortest Edge Length -The shortest edge length is a value that is input by the user. It determines the minimum allowable threshold for small features. It is used as an input to test for small curves, small surfaces, small volumes and close loops. The default value for this is 1. This value should be changed relative to the size of the model. In a very broad sense, it represents a desired mesh edge length. Curves and surfaces which are smaller than this size, and which may be troublesome to mesh with the desired granularity, will be flagged and they can be removed or modified.

Bad Angle Upper/Lower Bounds - The bad angle upper/lower bounds are tolerances set by the user to determine the definition of small or large angles. The default values are set at 350 degrees for the large angle and 10 degrees for the small angle. These values are used to test for angles between curves, surfaces, and at tangential intersections.

Bad Angle Check - The bad angle check will test for small angles between curves, surfaces, and at tangential intersections. The test will only look for curves or surfaces that are adjacent.

> **Tangential Intersection** - A tangential intersection is formed when two parallel surfaces share an edge and have a 180 degree angle between them. The tangential intersection test is looking for the condition where two surfaces that meet tangentially share a common edge, and each of the surfaces has another edge which resides on a third face and forms a small angle as shown in the following example. Surface 1 and Surface 2 are tangential to each other and share a common edge. Both Surface 1 and 2 have another edge which resides on Surface 3 and forms a small angle at the vertex common to all three surfaces.

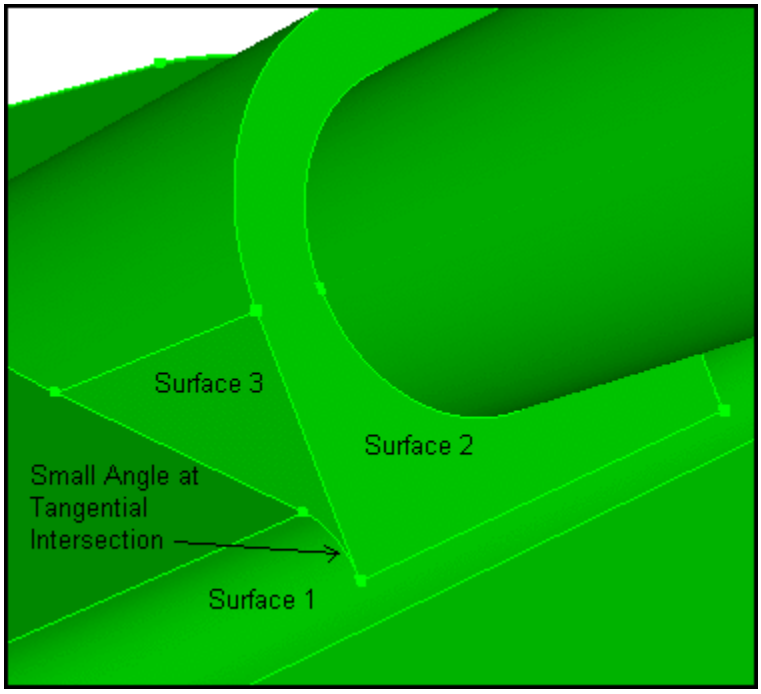


Figure 2. Tangential Intersection

Overlap Check - The overlap tests look for geometry that are either overlapping or coincident (exactly on top of each other). Keep in mind that some of these problems may disappear with imprinting and merging.

Small Features Check - Small features may be necessary and desirable in a model, but many times they are the result of poor geometry translation or import, or they may just not be important to the analysis. The small features tests look for small curves, small surfaces, and small volumes. These tests rely on the user-defined short edge length parameter. Small curves, including zero-length curves such as hardpoints, are compared directly against the defined parameter, and flagged if they less than or equal to the given parameter. Small surfaces and volumes, on the other hand, are compared against their hydraulic radius. For surfaces the hydraulic radius is **4*surface_area/perimeter**. For volumes the hydraulic radius is **6*volume/surface_area**.

Bad Geometry Definition Check - CUBIT uses third party libraries, such as ACIS from Spatial, Inc., or Granite from Parametric Technology Corporation, for much of its geometric modeling capabilities. The bad geometry definition check calls internal validation routines in these libraries, when available, to check for errors in geometry definition. If the third party library does not provide validation capabilities, this check will not return anything. Note: ACIS and Granite are trademarks of Spatial and PTC, respectively.

Blend Surface Check - A blend surface is a transition surface between two orthogonal planes, such as a fillet. The blend surface check identifies the surfaces which meet this criterion. Many times these surfaces are candidates for the split surface command or the remove surface command. The split surface command allows you to split these blend surfaces into two surfaces, making it easier to mesh the volume. The remove surface command removes the surface and extends the adjoining surfaces until they intersect.

Close Loops Check - Close loops (pronounced KLOS, not KLOZ) are two loops on a single surface for which the shortest distance between loops is less than a user specified tolerance. The tolerance for close loops is the square of the shortest edge length parameter. Close loops are common around holes and fillets, and are usually found where one loop is entirely within the other loop. These surfaces are often candidates for removal, or tweaking.

Mergeable Entities Check - As is suggests, this test is looking for entities that overlap and that can be merged. Pressing the "Merge all" button on the Power Tools will automatically merge all entities flagged by the merge test.

Geometry Repair Tools

Split Surface Button

The split surface tool is used to split a surface into two surfaces. This is useful for blend surfaces, for example, where splitting a surface may facilitate sweeping. To select a surface for splitting, click on the surface in the tree view. To select multiple surfaces in the window, hold the CTRL key while selecting surfaces (surfaces must be attached to each other). Then press the split surface button to bring up the Control Panel window with the ids of selected surfaces in the text input window. The split surface menu is located on the Control Panel under Geometry-Surface-Modify. You must press the Apply button for the command to be executed. You can also bring up the Split Surface menu by selecting surfaces in the tree view and selecting **Remove** from the right click menu.

Note: Pressing the any of the geometry tool buttons on the panel will only bring up applicable command panels on the Control Panel. You must press the Apply button on the Control Panel to execute the command.

Heal Button

The healing function in CUBIT is used to improve ACIS geometry that has been corrupted during file import due to differences in tolerances, or inherent limitations in the parent system. These errors may include: geometric errors in entities, gaps between entities, and the absence of connectivity information (topology). To heal a volume, select the volume in the geometry repair tree view. Then press the heal button. You may also press the heal button without a geometry selected in the window, and enter it later. The Control Panel window will come up under the Geometry-Volume-Modify option with the selected volume id highlighted. If no entity is selected, or if another entity type is selected, the input window will be blank. You can also open the healing control panel by selecting **Heal** from the right click menu in the geometry power tools window.

Tweak Button

The tweak command is used to eliminate gaps between entities or simplify geometry. The tweaking commands modify geometry by offsetting, replacing, or removing surfaces, and extending attached surfaces to fill in the gaps. Tweaking can be applied to surfaces, and it can also be applied to curves with a valence no more than 2 at each vertex. To tweak a surface, select the surface in the tree view. The Geometry-Surface-Modify control panel will appear with the selected surface id in the input window. Options for tweaking a surface from the GUI include: Replace, Offset, Move to Entity, Move to Location, Move Delta Distance, and Move Normal to Surface.

Tweaking is also available for curves. Tweaking a curve creates a blended or chamfered edge between two orthogonal surfaces. The curve option is located on the Geometry-Curve-Modify panel under the Blend/Chamfer pull-down option.

Note: Only curves with valence 2 or less at each vertex are candidates for tweaking. Any other curve will cause the Geometry-Surface-Modify menu to appear.

Merge Button

The merge command is used to merge coincident surfaces, curves, and vertices into a single entity to ensure that mesh topology is identical at intersections. Unlike other buttons on the geometry repair panel, the merge button acts as an "Apply" button itself. All geometry that is listed under "mergeable entities" will be merged.

Remove Button

The remove button is used to simplify geometry by removing unnecessary features. To use the remove feature, click on the surface(s) in the Tree View. Right click and select the Remove Option, or click the Remove icon on the toolbar. The Control Geometry-Surface-Modify control panel will appear, with the surface ids in the input window. The Remove control panel can also be accessed from the right-click menu in the Geometry Power Tools window. Select options and press apply.

Composite Button

The composite button is used to combine adjacent surfaces or curves together using virtual geometry . Virtual geometry is a geometry module built on top of the ACIS representation. Surfaces may be composited to simplify geometry in order to facilitate sweeping and mapping algorithms by removing constraints on node placement. It is important to note that solid model operations such as webcut, imprint, or booleans, cannot be applied to models that have virtual geometry. Both curves and surfaces may be composited.

Collapse Button

The collapse angle button uses virtual geometry to collapse small angles. This is accomplished by partitioning and compositing surfaces in a way so that the small angle gets merged into a larger angle. Pressing the collapse button on the geometry power tools will open the collapse menu under Geometry-Vertex-Modify control panel. This panel can also be opened by selecting **Collapse** from the right click menu in the Geometry Tools window.

Reset Graphics Button

The reset graphics button will refresh the graphics window display.

Right Click Menu

The following right click menu is available from the geometry power tools. Specific options depend on the type of entity selected.

- **Zoom To**- Zoom to selected entity in the graphics window
- **Reset Zoom** - Reset graphics window zoom
- **Fly-in** - Animated zoom
- **Locate** - Labels the selected entity in the graphics window
- **Draw** - Draw this entity by itself
- **Draw with Neighbors** - Draw this entity with all attached neighbors
- **Clear Highlights** - Clear all highlighted entities and reset graphics
- **Reset Graphics** - Reset graphics window
- **Tweak** - Opens the tweak menu in the main control panel
- **Remove** - Opens the remove menu in the main control panel
- **Split** - Opens the split surface or split curve menu in the main control panel, depending on the type of entity selected.
- **Merge Selected** - Merge selected entity from mergeable entities list
- **Merge All** - Merge all entities listed in the mergeable entities list
- **(Virtual) Composite** - Opens the composite menu in the main control panel
- **(Virtual) Collapse** - Opens the collapse menu the main control panel

Meshing Tools

The meshing power tool provides a tool for determining whether a geometry can be meshed automatically, or if it requires further geometry modification. This tool is designed to help guide users through geometry decomposition process by providing a convenient way to see which geometries need further modification or decomposition prior to meshing.

Figure 1. Meshing Power Tools

Steps to Using Meshing Power Tool

1. Select entity type
2. Select entity ID
3. Press analyze
4. View results in result window
5. Toggle visibility in graphics window by pushing the purple refresh button. Red entities are not meshable; Green entities are ready to mesh.

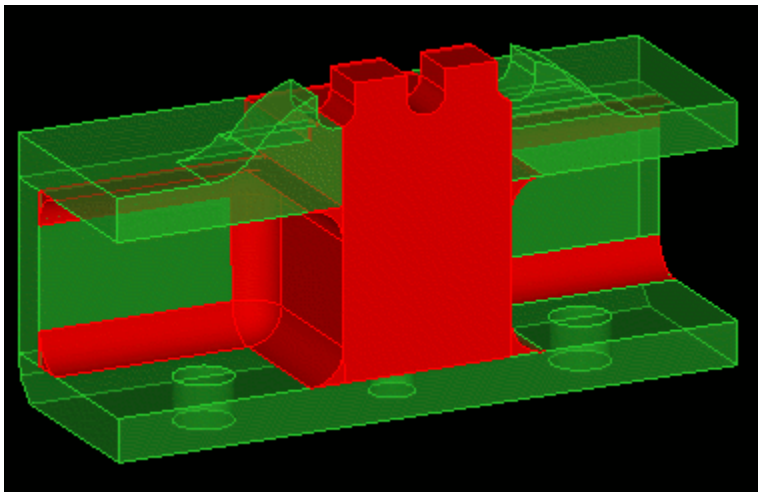


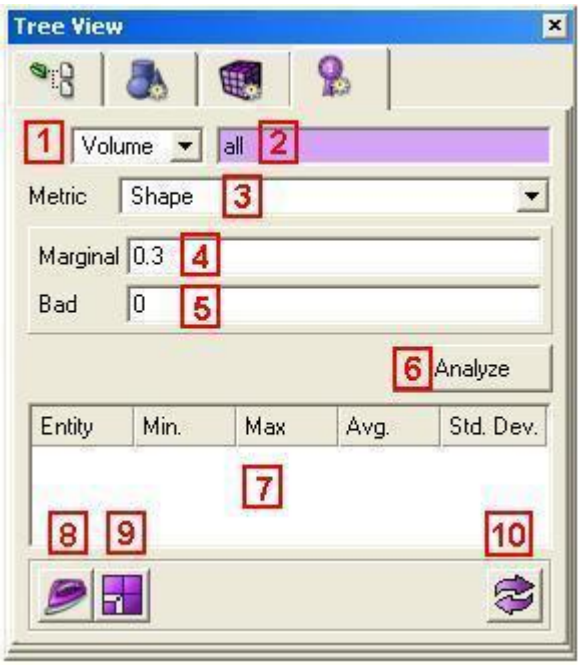
Figure 1. An example geometry that has been analyzed with the mesh tools. Green volumes have had their scheme set. Red volumes require further decomposition

Right Click Context Menu

- **Zoom To** - Zoom in on this element in the graphics window
- **Draw** - Draw this entity by itself in the graphics window
- **Visibility On/Off** - Toggle visibility
- **Reset Graphics**- Reset graphics display
- **View Descendants** - Opens a list of child entities and their meshing schemes
- **View Ancestors**- Open a list of parent entities and their meshing schemes
- **View Neighbors**- Open a list of bordering entities and their meshing schemes

Mesh Quality Tools

The Mesh Quality Tool is located in the Entity Tree window under the quality tab. The Mesh Quality Tool is used to analyze mesh quality, isolate bad elements, and improve quality through smoothing or refinement. The following image shows the mesh quality tool. Each of the numbered items is described below.



1. **Entity Type** - The entity types are selected from the pulldown menu. The following entity types are available: Volume, Surface, Curve, Vertex, Quad, Tri, Hex, Tet, Node, Edge. Quality metrics are not available for all entity types (e.g. Zero-dimensional or 1-dimensional entities such as curves, edges, vertices and nodes). These entities are only included on the mesh quality tool because they can be used for refinement.
2. **ID Range** - IDs may be entered by any of the typical methods described in Control Panel Functionality . Selecting all will analyze all elements of that type.
3. **Quality Metric** - There are a wide range of quality metrics available for analyzing mesh quality. The quality metrics are described in detail in the Mesh Quality section. The default quality metric is the Shape metric.
4. **Marginal Element Threshold** - The threshold for marginal elements. A higher number will typically increase the number of elements that fall into marginal and bad ranges. Setting the threshold to zero will make all elements fall into the good range.
5. **Bad Element Threshold** - The threshold for bad elements. Increase this number if you require a better quality mesh.
6. **Analyze Button** - Analyzes the selected elements for the quality metric given.
7. **Output Window** - The output window for results of mesh quality analysis. Entities will be grouped by quality ranges: good, marginal, and bad. The columns give information regarding quality of elements in the mesh. The Entity column lists the entity id. Min and Max give the ids and values of the best and worst elements. Avg. determines the average quality of all elements in the range. Std. Dev calculates the standard deviation.
8. **Smooth Button** - Smoothing is a method of improving mesh quality by moving nodes until elements achieve a desirable shape. Pressing the smooth button will open the Smooth menu on the Control Panel. Several smoothing schemes are available. See Mesh Smoothing for descriptions of smoothing algorithms.
9. **Refine Button** - The Refine button is used to increase mesh density at or around the particular entity. Pressing the refine button will open the Refine menu on the Control Panel. Several options for refinement are available. See Mesh Refinement for more information.
10. **Reset Graphics Button** - Used to reset the graphics display

Property Editor

The Property Editor is a window that lists properties about the current entity selection. Some of the properties, like CUBIT ID, entity type, or geometry engine, are listed for reference only. Other attributes, like name, or mesh intervals, color, mesh scheme, or smooth scheme can be edited from the window. The Property Editor is located on the left panel in the GUI. The highlighted entity/entities in the graphics window are listed in the property editor window. The Property Editor also lists information about selected mesh entities, boundary conditions, and assemblies. Selecting an object from the Tree View will also open the object in the property editor.

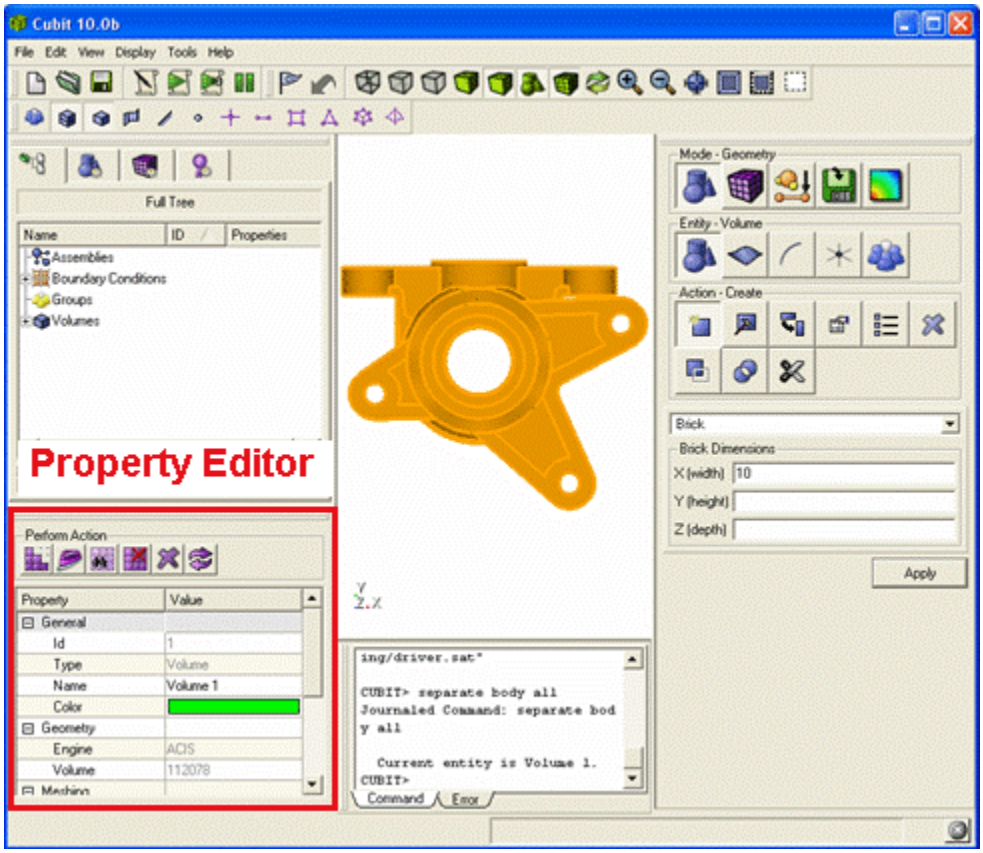


Figure 1. Property Editor Window

The row of buttons on the top of the editor are shortcuts to common commands. These include:

Meshes the selected entity/entities at their current interval and scheme settings

Smooth selected entity using the current smoothing scheme

Preview mesh intervals on selected entity

Delete mesh on specified entity (do not propagate to lower order entities)

Delete current entity

Reset entity to default settings and delete mesh

Editing Entity Attributes from the Property Editor

The Property Editor provides a convenient way to change attributes on entities. The rules for editing fields in the Property Editor are given below. Some of the fields cannot be changed, some can be edited from an input field, and others are edited by selecting from a list, or by opening the corresponding window from the Control Panel. This is not an exhaustive list of attributes; it is meant to show the different types of input fields.

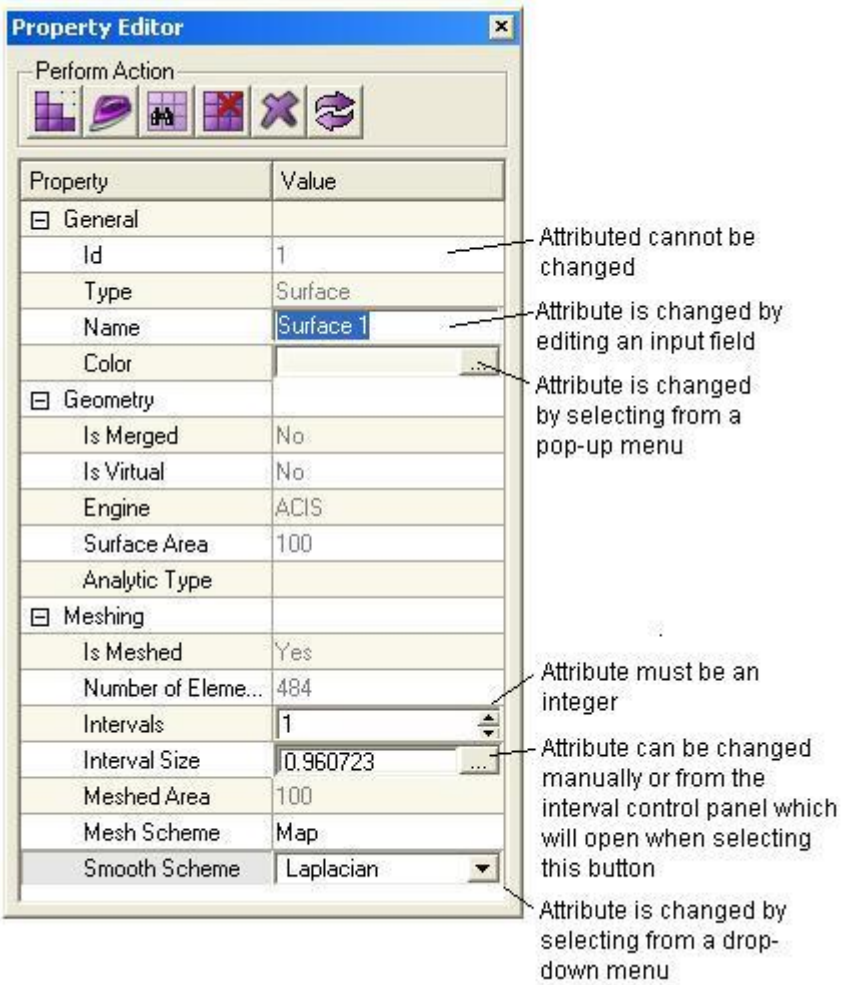


Figure 2. Editing Attributes in the Property Window

Below is a summary of properties listed for each selection type. Red attributes are ones that can be changed from the property editor.

Volume - id, type, *name, color*, geometry engine, volume of volume, is_meshed, number of elements, intervals, interval size, meshed volume, *mesh scheme, smooth scheme*

Surface - id, type, *name, color*, is_merged, is_virtual, is_meshed, geometry engine, surface area, analytic type (cone, sphere, etc), is_meshed, number of elements, *intervals, interval size*, meshed area, *mesh scheme, smooth scheme*

Curve - id, type, *name, color*, is_merged, is_virtual, engine, length of curve, analytic type, is_meshed, *intervals, interval size*, length of meshed edges, *mesh scheme*

Vertex - id, type, *name, color*, is_merged, is_virtual, location, is_meshed

Hex/Tet - id, exodus id, geometry owner, quality, contained node ids

Face/Tri - id, exodus id, geometry owner, quality, contained node ids

Mesh Edge - id, exodus id, geometry owner, contained node ids

Node - id, exodus id, geometry owner, location

Block - id, *element type*, element count, *attribute count*, *attributes* (up to 10)

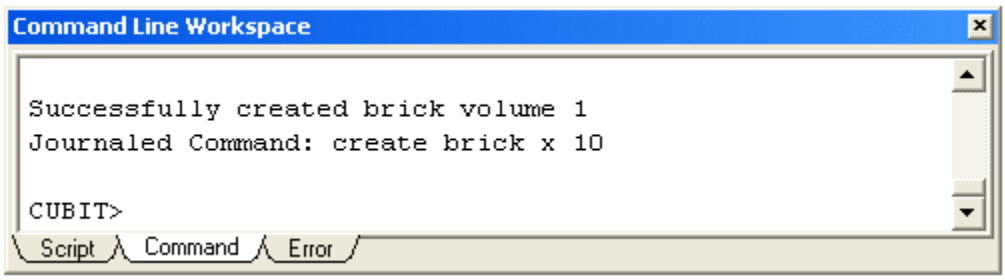
Sideset - id, element count

Nodeset - id, node count

Part - name, description, material description, material specification, file, units, instance

Assembly - classification level, classification category, weapon category

Command Line Workspace



The Command Line Workspace is the interface for command interaction between the user and the CUBIT application. The user can enter commands into this window as if they were using the command line version of CUBIT. Journaled commands will be echoed to this screen, even if they were not typed in manually. Thus, if the user wants to know what the command sequence for a particular action on the GUI is, they can watch for the "Journaled Command:" line to appear. In addition, this screen will contain important informational and error messages. The command window has the following three tabs:

1. Command
2. Error
3. Script

Command Window

The command line workspace emulates the environment in the command line version of CUBIT. Commands can be entered directly by typing at the **CUBIT>** prompt. This window also prints out error messages, informational messages, and journaled commands.

Entering Commands

To enter commands in the command line workspace, the command window must be active. Activate the command window by clicking anywhere inside the window. Commands are typed in at the **CUBIT>** prompt. If you do not remember the specific command sequence you can type **help** and the name of the command phrase. The input window will show all of the commands that contain that word or phrase. Alternatively, if you know how a command starts, but do not remember all of the options, you can type **?** at the end of the command to show all possible command completions. See Command Syntax for an explanation of command syntax rules.

Repeating Commands

Use the **Up** and **Down** arrow keys on the keyboard to recall previously executed commands.

Commands can be repeated in other ways as well.

- Hitting the enter key while the cursor is on a previous command line will copy that command to the current prompt.
- The command window supports copy and paste for repeating commands.
- You can use drag and drop to copy a previous command to the current command prompt. To do so, select the desired text and drag it to the current command prompt.

Error Window

The error window is located in the Command Line Workspace under the Error tab. If there are errors, a warning icon will appear on the tab. The icon will disappear when you open the window to view errors. The error window only displays the error output, which can make it easier to find and read the error output. The command that caused the error will be printed along with the error information. If the command was from a journal file, the file name and number will be printed next to the command.

Script Window

CUBIT boasts a robust Python interpreter built right into the graphical user interface. To create a Python script using the Script tab, start typing at the "%>" prompt. At the end of each line, hit **Enter** to move to the next line . To execute the script, press **Enter** at a blank line. Scripts may also be written in the Journal File Editor.

The interface between cubit and python is the "cubit" object. This object has a method called **cmd** which takes as an argument a command string. Thus, the following command in the script window:

```
cubit.cmd("create brick x 10")
```

will create a cube with sides 10 units long. The following script is a simple example that illustrates using loops, strings, and integers in Python.

```
%>for i in range(4):
. . x=i*3
. . for j in range(4):
. .   y=j*3
. .   for k in range(4):
. .     z=k*3
. .     mystr="create vertex x " + str(x) " y " + str(y) + " z " + str(z)
. .     cubit.cmd(mystr)
```

This simple script will create a grid of vertices four wide. Scripts can be more advanced, even creating customized windows and toolbars. If you would like more information on advanced scripting with CUBIT, please contact the design team.

Docking and Undocking the Input Window

The command window can be undocked by clicking and dragging the left edge. If it is floating it can be redocked by double-clicking the solid blue bar. By default, it will always be redocked in the bottom left corner of the application window. To change the size of the floating window, click and drag the edge of the window. To change the height of the docked window, click and drag the top edge or right edge.

Journal File Editor

The Journal File Editor is a built-in text editor that can read, edit, play, and translate CUBIT journal files and Python Scripts. To open the journal file editor select the icon on the File Tools toolbar, or from the Tools Menu.

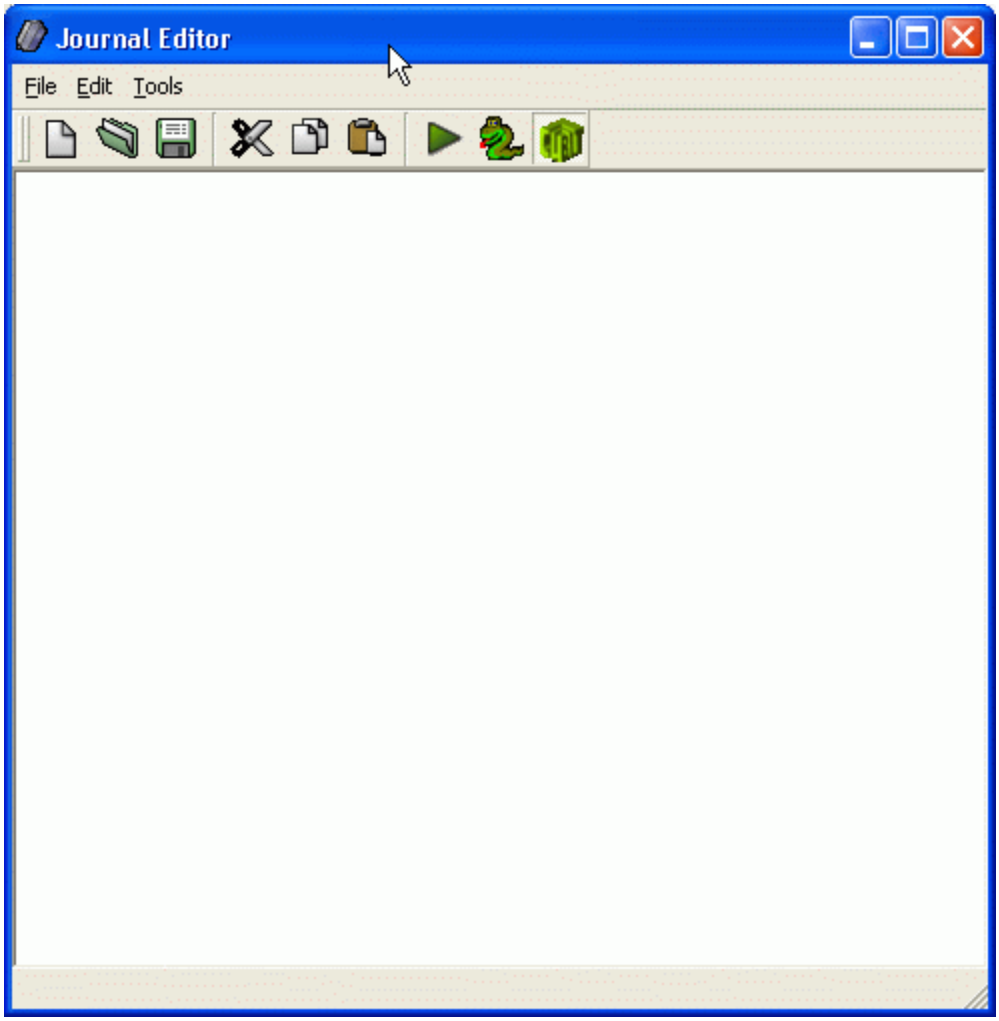


Figure 1. The Journal File Editor

The Journal File Editor allows users to enter Cubit commands then execute them incrementally. The resulting journal file may be saved and used later. The Journal Editor can accept standard Cubit commands as well as Python scripts. The Journal Editor can translate between the two forms.

Journal Editor Toolbar

The Journal Editor's Toolbar provides quick access to several important functions.

- **New** - Clears current edit space and prepares for a new Journal File
- **Open** - Shows standard file open dialog
- **Save** - Shows standard file save dialog
- **Cut** - Standard text cut operation
- **Copy** - Standard text copy operation
- **Paste** - Standard text paste operation
- **Play Journal File** - Plays the entire journal file
- **Translate to Python** - Translates the current Cubit commands in the journal file to Python scripts.

- **Translate to Cubit** - Translates the current Python script in the journal file to Cubit commands.

Right Click Menu

- **Play Selected** - Play the selected lines.
- **Undo** - Undo the last menu operation
- **Redo** - Redo the last operation, after Undo.
- **Cut** - Cut the current selection and place it on the clip board.
- **Copy** - Copy the current selection to the clip board.
- **Paste** - Paste clipboard text at the cursor.
- **Select All** - Select all text in the Journal Editor

Toolbars

The CUBIT toolbars provide an effective way for accessing frequently used commands.

Below is a brief description of each of the available toolbars. To view a description of the function of each tool, hold the mouse over the tool in the CUBIT Application to display tool tips.

File

Provides CUBIT (*.cub) file operations. This toolbar also includes Journal File operations.

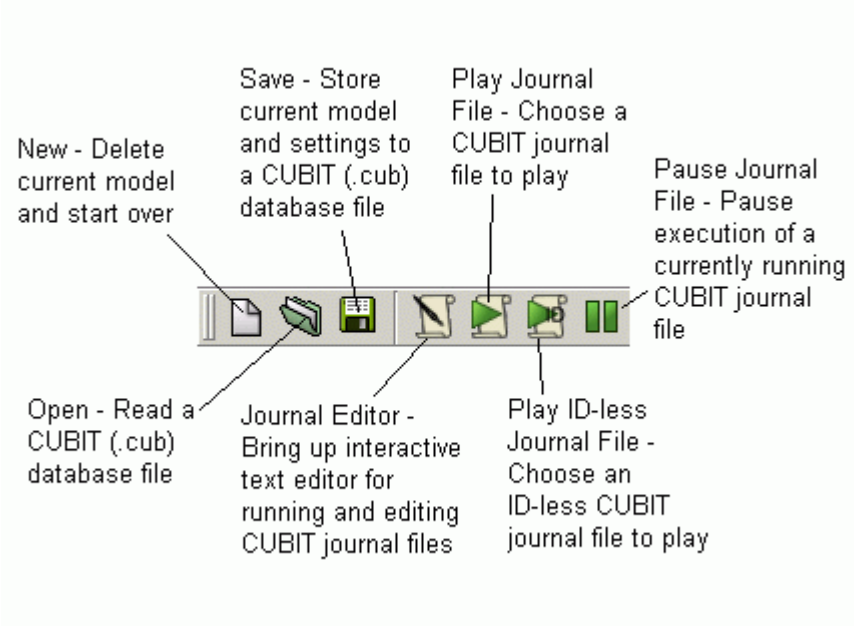


Figure 1. File Toolbar

Display

Controls the display mode in the Graphics Window.

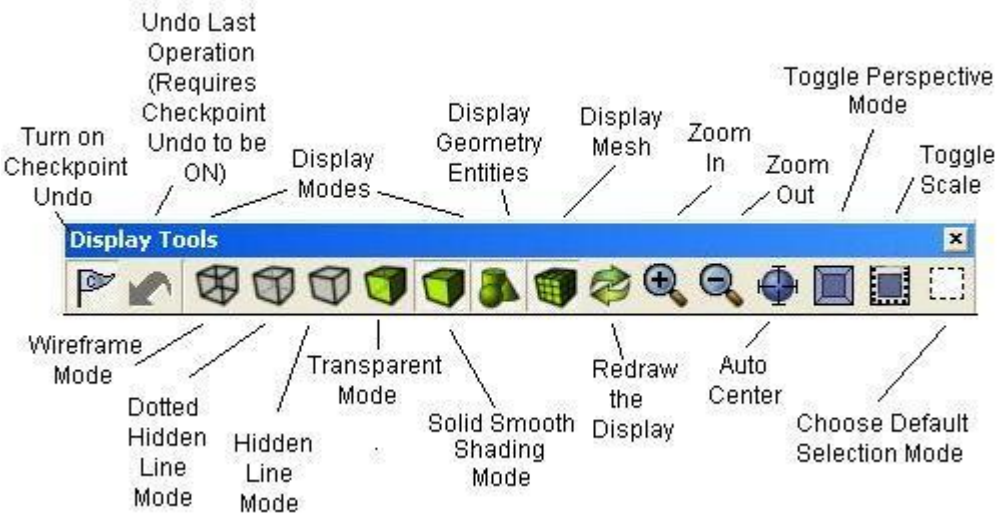


Figure 2. Display Toolbar

Select

Controls the Entity Selection Mode for picking or selecting entities.

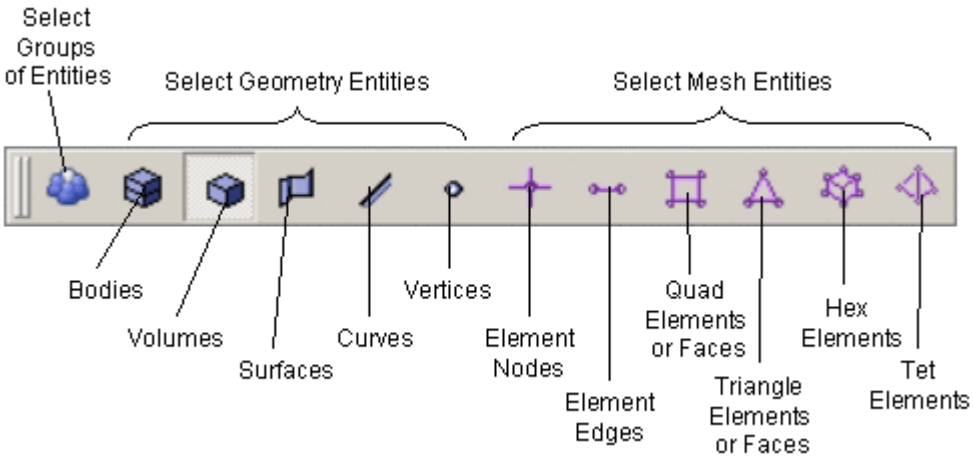


Figure 3. Select Toolbar

Drop Down Menus

The CUBIT Drop-Down Menus, located at the top of the CUBIT Application Window provide access to capabilities such as file management, checkpoints, display manipulation, journaling, system setup, component management, window management, and help.

File

This menu provides common file operations, including Importing and exporting of files. A list of recently saved or imported files is also provided, allowing a quick way to import current or recent work. Users can also exit and reset the program from this menu. For more on importing files, see Importing Files from the GUI.

Edit

This menu only provides a way to enable the Checkpoint feature of the system. If Checkpoint is enabled, one level of Undo is available to the user.

View

The View Menu lists all available toolbars and windows in the current CUBIT session. Selecting a toolbar or window will make it visible. Deselecting a toolbar or window will hide it. You can also hide an undocked window or toolbar by clicking on the small "x" in the upper right corner. For more information on docking and undocking toolbars, see CUBIT Application Window.

Display

The Display Menu controls display options for the graphics window. These options are explained below:

- **View Point** - Controls the camera view point. Toggles between front, top, right, and isometric views.
- **Render Mode** - Controls visibility modes, including: wireframe, true hidden, hidden line, transparent, and shaded.
- **Geometry** - Controls geometry visibility
- **Mesh** - Controls mesh visibility
- **Refresh** - Updates the graphics display
- **Background** - Changes the background color
- **Zoom In** - Enlarges the model in graphics window
- **Zoom Out** -Shrinks the model in graphics window
- **Zoom To Fit** - Enlarges or shrinks model in the graphics window so it fills the whole screen
- **Toggle Perspective** - When this option is selected, the entities in the graphics display window are drawn in perspective mode.

Tools

The Tools Menu contains access to GUI-specific tools and options. These options are explained below.

- **Journal Editor** - Opens journal file editor. The Journal Editor is used to write, edit, play, and save journal files. It can also be used to create and manage Python scripts. A built-in translator will convert between the two files types.
- **Play Journal File** - Plays a specified journal file. You can browse through files and folders on your computer to select the journal file to play.
- **Options** - Opens the Option dialog box. This dialog box controls all of the preferences for the GUI including display colors and widths, mouse settings, journal file options, mesh and geometry defaults, and general layout preferences.
- **Components** - Opens the Components dialog box. This window is used to load and unload external and internal components.

Help

Menu to bring up on-line searchable documentation on most features in CUBIT (this document). Also provides the current version number and copyright information.

Importing Files from the GUI

To import a geometry file, go to the File Menu and select Import. Choose the file type, select the file, and press Open. The following file types are available. Each file type has different import options, which are described below.

- Importing ACIS/IGES/STEP Files

- Importing Exodus II Files
- Importing Facet/AVS/STL Files
- Importing CUB Files

Importing ACIS/IGES/STEP Files

- **Ignore Bodies** - Bodies will not be imported
- **Ignore Free Surfaces**- Free surfaces will not be imported
- **Ignore Free Curves**- Free curves will not be imported
- **Ignore Free Vertices**- Free vertices will not be imported
- **Heal on Import**- This feature is only available for IGES and STEP files. Autoheal will be run on imported bodies.
- **Convert Free Surfaces to Bodies**- This feature is only available for IGES files. Converts free standing surfaces into bodies.
- **Show Each While Importing**- Shows each body individually while it is being imported
- **Separate Multi-Volume Bodies**- This feature will separate multi-volume bodies into multiple bodies. This button is checked by default.
- **Create Group**- Creates a group for the imported geometry. Type the name of the group in the window.
- **Import XML**- Reads assembly information and other metadata from an XML file in the DART metadata XML format.

Importing Exodus II Files

There are two main methods for importing Exodus II mesh data files into CUBIT. The first method is to import the mesh and construct Mesh Based Geometry. The second method is to import the mesh by itself, or associate the mesh with geometry that already exists in CUBIT.

1. Importing Mesh Geometry

- **Create Geometry from Blocks**- This option will allow the user to specify individual blocks to be read in from the file. If the box is NOT checked, all blocks will be imported. If the box is checked, the user may select specific blocks to import. The pulldown menu will show a list of blocks in the file. Standard parsing is also available in this input window (i.e. 1 to 5). Each block will define a new body in the mesh-based geometry.
- **Create Geometry from Nodesets/Sidesets**- This option enables the program to use any nodeset or sideset information stored in the Exodus file to define surface and curve boundaries in the mesh-based geometry. It is always checked by default.
- **Feature Angle** - The feature angle defines the maximum angle allowed between facets on a surface before it is split by a curve, and the maximum angle on a curve before it is split by a vertex. The default angle is 135 degrees.
- **Smooth Curves/Surfaces** - The smooth curves option allows the use of a higher order approximation to the surface instead using the original mesh edges to constrain the boundary.
- **Merge Geometry** - Use this option to merge resulting volumes. Merge Nodes - The merge nodes option will merge nodes that are close to each other. The Node Merge Tolerance defines the maximum allowable distance between nodes, and can be changed by the user.
- **Apply Deformations** - If time dependant data is available in the Exodus II file, the user may select a specific time, or choose from a list of time steps. The default time step is the last.

2. Importing a Free Mesh

CUBIT also has the capability to import a free mesh by itself, or to import a free mesh and associate it with geometry already in CUBIT. If it is associated with geometry already in CUBIT, the geometry must be the same geometry from which the mesh was exported. Options for importing free meshes are explained below.

- **Blocks**- The Block check box is used to define a specific block to import. If the box is not checked, all blocks will be read in by default. If the box is checked, enter the ID of the block to import, or enter a range of blocks if multiple blocks are desired.

- **Specify Geometry ID/Range** - This is another method of specifying only a portion of the mesh to be read in from the file. Select the IDs of current geometry to indicate which portions of the mesh to import. This option is only available when associating mesh with geometry that already exists in CUBIT
- **Unique Genesis IDs**- A flag to say whether imported genesis entities keep thier ids, or get new ones in case of id clashing.
- **Shell** - Check this flag if there are shell elements in the model. The import code cannot always detect them.

Importing Facet/AVS/STL Files

Facet files are files composed of triangles that have been stitched together to form surfaces. CUBIT supports three types of facet file extensions: ".fac", ".avs" and ".stl". For more information on facet files see Importing Facet Files. Options for importing facet files are explained below.

- **Feature Angle** - Defines maximum allowable angle between facets on a given surface. Surfaces with angles greater than this will be split into multiple surfaces. An angle of 180 will create a surface for every facet. An angle of 0 will create a single surface.
- **Smooth Curves/Surface** - Creates a higher order approximation to the surface to smooth out the boundary elements.
- **Merge Geometry** - Merges coincident bodies on import
- **Merge Nodes** - Merges coincident nodes that are within the specified Node Merge Tolerance
- **Make Elements** - Creates triangular elements out of the facets
- **Stitch** - The stitch option is used with the facet or avs format files to try to merge vertices and triangles that are close in order to make a water tight model.

Importing CUB Files

Files with the .cub extension are the cross-platform, binary Cubit project files that are saved by default when selecting Save from the File menu. These files consist of geometry and the associated mesh, groups, blocks, sidesets, and nodesets. Mesh and geometry are restored from the Cubit file in exactly the same state as when saved. There are two methods of restoring .cub files. The first method is by selecting Open from the File Menu. The second method is to select Import from the file menu, and set the import type to .cub. The first method will close the current session, and open the model in a new session. The second method will open the model on top of other models in the current session.

Options Menu

To change program preferences in the Graphical User Interface select: **Tools > Options** . The options menu includes:

- Display
- General
- Geometry Defaults
- History and Cubit Journalling
- Layout
- Mesh Defaults
- Mouse Settings

Display Preferences

This menu controls entity display features for the graphics window which include the following:

- Curve Line Width

- Highlight Line Width
- Vertex Size
- Node Size
- Text Size
- Background Color
- Graphics Shrink Percentage

General Preferences

This menu controls general program options including the following:

- Prompt for Unsaved Application Data
- Prompt for Unsaved Journal Data
- Change to Script Directory
- Show Startup Splash Screen
- Show Tip of the Day
- Post Processor Executable Directory

Geometry Defaults

This menu controls the geometry defaults. The user can select between:

- ACIS
- Facets
- Pro Engineer/Granite

History Preferences

This menu controls the input window history and journal file options. These include:

- Maximum Number of Commands - The max number of commands kept in the current command history.
- Comment Line Filtering - Whether to count comments in command history.
- Maximum Number of Lines - Maximum number of lines in input window.
- Number of Slack Lines - Number of consecutive blank lines to allow
- Journal Command History - Whether to use a journal file to save command history. Default is to use a journal file.
- Journal File Directory - Where the journal file will be saved. Default is the starting directory.
- Journal File Name - The name of the journal file. A name will be given by default if one is not specified.

Layout Preferences

This menu option controls input window formatting and control panel docking options. These features are exclusive to the GUI.

- Font for command line workspace
- Font size for command line workspace
- Reset Window Layout Button - Used to reset GUI windows to their default positions

Also included in the layout preferences is a list of available windows a checkbox to show/hide each window.

Buttons

The Buttons menu controls the look and feel of buttons on the GUI.

- Use Labels on Buttons
- Preferred Location (currently under construction)

Cubit Layout Settings

- Show script tab - Shows the script tab on the input window

Mesh Defaults

- Hex/Quad
- Tet/Tri

Mouse Settings

This menu controls mouse button controls. Pressing the **Emulate Command Line Settings** button will cause all of the settings to simulate mouse controls in the command line version of CUBIT. For a detailed description of mouse settings see the **View Navigation-GUI** page.

Command Recording and Playback

Sequences of CUBIT commands can be recorded and used as a means to control CUBIT from ASCII text files. Command or "journal" files can be created within CUBIT, or can be created and edited directly by the user outside CUBIT.

- Journal File Creation & Playback
- Controlling Playback of Journal Files
- Automatic Journal File Creation
- IDless Journal Files

Journal File Creation and Playback

Recording a Session

Command sequences can be written to a text file, either directly from CUBIT or using a text editor. CUBIT commands can be read directly from a file at any time during CUBIT execution, or can be used to run CUBIT in batch mode. To begin and end writing commands to a file from within CUBIT, use the command

```
Record '<filename>'

Record Stop
```

Once initiated, all commands are copied to this file after their successful execution in CUBIT.

Replaying a Session

To replay a journal file, issue the command

```
Playback '<filename>'
```

Journal files are most commonly created by recording commands from an interactive CUBIT session, but can also be created using automatic journaling or even by editing an ASCII text file.

Commands being read from a file can represent either the entire set of commands for a particular session, or can represent a subset of commands the user wishes to execute repeatedly.

Two other commands are useful for controlling playback of CUBIT commands from journal files. **Playback** from a journal file can be terminated by placing the **Stop** command after the last command to be executed; this causes CUBIT to stop reading commands from the current journal file. Playback can be paused using the Pause command; the user is prompted to hit a key, after which playback is resumed.

Journal files are most useful for running CUBIT in batch mode, often in combination with the parameterization available through the APREPRO capability in CUBIT. Journal files are also useful when a new finite element model is being built, by saving a set of initialization commands then iteratively testing different meshing strategies after playing that initialization file.

Controlling Playback of Journal Files

The following commands control the playback of Journal Files:

stop
pause
resume [<n>]
where
next [<n>]

The playback of a journal file can be interrupted in two ways. Pressing **ctrl-c** while the journal file is playing will halt playback of the journal file (Interrupting Running Tasks). Alternately, if the **stop** or **pause** commands are encountered in the journal file and CUBIT is reading commands from a terminal (as opposed to a redirected file), playback of the journal file will halt after that command. When playback of a journal file is interrupted, you will be presented with a command prompt.

Command Line Behaviors

In the command line version of CUBIT you can resume playback of a journal file with the **resume** command. If playback was interrupted because **ctrl-c** was pressed, it will resume at the next command after the one that was interrupted. If playback stopped because of a **stop** or **pause** command in the journal file, it will resume at the next line after the **stop** or **pause** command.

If journal files that are playing back contain **playback** commands themselves, there may be multiple current journal files. The **where** lists all current journal files and where the journal files have paused. Each line contains the stack position (a number), the filename and the current line in the file. Unless CUBIT is running in batch mode, the first line is always **<stdin>**. This just means that CUBIT will return to the command prompt after the top-most journal file has completed.

The remaining portion of any active journal file may be skipped by specifying the stack position (first number on each line of the output from the **where** command) of the file where you want to resume. Any remaining commands in active journal files with lower stack positions will be skipped.

The **next** command steps through interrupted journal files line-by-line. The argument to the **next** command is the number of lines to read before halting playback again. If no number is specified, the command will advance one line.

Automatic Journal File Creation

Controlling Automatic Journal File Creation

By default, CUBIT automatically creates a journal file each time it is executed. The file is created in the current directory, and its name begins with the word " cubit " followed by a number between 01 and 992, e.g. cubit01.jou . Journal file names end with a ".jou" extension, though this is not strictly required for user-generated journal files. If no journaling is desired, the user may start CUBIT with the -nojournal command line option or use the command :

[set] Journal {Off | On}

Turning journaling back on resumes writing commands to the same journal file3.

Most CUBIT commands entered during a session are journaled; the exceptions are commands that require interactive input (such as Zoom Cursor), some graphics related commands, and the **Playback** command.

Recording Graphics Commands

All graphics related commands may be enabled or disabled with the command:

Journal Graphics {On | Off}

The default is **Journal Graphics Off** .

Recording Entity IDs and Names

When an entity is specified in a command using its name, the command may be journaled using the entity name, or by using the corresponding entity type and id. The method used to journal commands using names is determined with the command:

Journal Names {On | Off}

The default is **Journal Names On** .

If an entity is referred to using its entity type and id, the command will be journaled with the entity type and id, even if the entity has been named.

Recording APREPRO Commands

APREPRO commands may be echoed to the journal file using the following command

Set aprepro [ON|off]

See APREPRO Journaling for more information.

Recording Errors

The default mode for CUBIT is to not journal any command that does not execute successfully. To turn this mode off and echo all commands to the journal file, regardless of the success status, use the following command:

Journal Errors {on|OFF}

If a command did not execute successfully and the journal errors status is ON, then the unsuccessful command will be written as a comment to the file. For example an unsuccessful command might look like the following in the journal file

create brick x 10 x 10 z 10

Since CUBIT recognizes this as erroneous syntax, it will issue an error when the command is issued, but will still write the command to the journal file as a comment, prefixing the command with "##".

This option may be useful when tracking or documenting program errors.

Idless Journal Files

Journal files can also be created without reference to entity IDs. The purpose of this command is to enable journal files created in earlier versions of CUBIT to be played back in newer versions of CUBIT. Using the "IDless" method, commands entered with an entity ID will be journalled with an alternative way of referring to the entity. Changes in CUBIT or ACIS often lead to changes in entity IDs. For example, a webcut may result in volume 3 on the left and volume 4 on the right. In another version of CUBIT, those entity IDs may be swapped (4 on the left and 3 on the right). Playing an IDless journal file makes the actual ID of an entity irrelevant.The syntax for this command is:

[set] Journal IDless {on|off|reverse}

The **on** option will enable idless journalling, and commands will be journalled without entity IDs. For example, "mesh volume 1" may be journalled as "mesh volume at 3.42 5.66 6.32 ordinal 2".

Selecting the **off** option will cause commands to be journalled in the traditional manner (i.e., as they are entered).

The **reverse** option allows you to convert idless journal files back into an ID-based journal file where the new journal file will reflect current numbering standards for IDs.

If you issue the command **Journal IDs** without any additional options, then the current status of ID journalling is printed. At startup, this should be "off".

The most likely scenario for converting older journal is to use the record command during playback. The following is an example.

```
journal idless on
record "my_idless.jou"
playback "my_journal.jou"
record stop
journal idless off
```

To record an *idless* journal file back into an *id-based* journal file you might use the following sequence.

```
journal idless reverse
record "new_id_based.jou"
playback "my_idless.jou"
record stop
journal idless off
```

Graphics Window Control

The graphics display windows present a graphical representation of the geometry and/or the mesh. The quality and speed of rendering the graphics, the visibility, location and orientation of objects in the window, and the labeling of entities, among other things, can all be controlled by the user.

Unless the **-nographics** option was entered on the command line, a graphics window with a black background and an axis triad will appear when CUBIT is first launched. The geometry and mesh will appear in this window, and can be viewed from various camera positions and drawn in various modes (wire frame, hidden line, smooth shade, etc.). This section will discuss methods for manipulating the graphics with the mouse and for controlling the appearance of entities drawn in the graphics window.

Graphics in CUBIT operates on the principle of a "display list", which keeps track of various entities known to the graphics. All geometry and mesh objects created in CUBIT are put into the display list automatically. The visibility and various other attributes of entities in the display list can be controlled individually. In addition, CUBIT can also optionally display entities in a temporary mode, independent of their visibility in the display list. Drawing of items in temporary mode can be combined with the display list to customize the appearance. The overall display is controlled by various attributes like graphics mode, camera position, and lighting, to further enhance the graphics functionality.

The following items discuss the various graphics capabilities available in CUBIT:

Command Line View Navigation: Rotate Zoom and Pan

Mouse Based View Navigation: Rotate Zoom and Pan

Updating the Display

Graphics Modes

Drawing and Highlighting Entities

Mesh Slicing

Entity Labels

Colors

Geometry and Mesh Entity Visibility

Graphics Camera

Graphics Lighting Model

Graphics Window Size and Position

Saving Graphics Views

Hardcopy Output

Miscellaneous Graphics Options

Updating the Display

Among the most common graphics-related commands is **Display** . This command clears all highlighting and temporary drawing, and then redraws the model according to the current graphics settings. Two related commands are **Graphics Flush** , which redraws the graphics without clearing highlighting or temporary drawing. **Graphics Flush** is useful when a previously executed command modified the graphics and didn't update the screen and the user wishes to update the display. Also, **Graphics Clear**, which clears the graphics window without redrawing the scene, leaving the window blank.

NOTE: Although most changes to the model are immediately reflected in the graphics display, some are not (for graphics efficiency). Typing **Display** will update the display after such commands. **Ctrl-R** will also update the display as long as the mouse is in the graphics window.

Prevent Graphics From Updating

For especially large models, it may take excessively long to update the display after an action has been performed. To prevent the graphics from automatically updating, use the following command:

graphics pause

This command prevents the graphics window from being updated until the next time the **Display** command is issued**.**

NOTE: The **Plot** command is synonymous to the **Display** command, and either can be used with identical results**.**

Command Line View Navigation: Zoom, Pan and Rotate

Commands used to affect camera position or other functions are listed below. All rotation, panning, and zooming operations can include the **Animation Steps** qualifier, makes the image pass smoothly through the total transformation. Animation also allows the user to see how a transformation command arrives at its destination by showing the intermediate positions.

Rotation

Rotate <degrees> About [Screen | Camera | World] {X | Y | Z} [Animation Steps <number_steps>]

Rotate <degrees> About Curve <curve> [Animation Steps <number_steps>]

Rotate <degrees> About Vertex <vertex_1> Vertex <vertex_2> [Animation Steps <number_steps>]

Rotation of the view can be specified by an angle about an axis in model coordinates, about the camera's "At" point, or about the camera itself. Additionally rotations can be specified about any general axis by specifying start and end points to define the general vector. The right hand rule is used in all rotations.

Plain degree rotations are in the Screen coordinate system by default, which is centered on the camera's At point. The **Camera** keyword causes the camera to rotate about itself (the camera's From point). The **World** keyword causes the rotation to occur about the model's coordinate system. Rotations can also be performed about the line joining the two end vertices of a curve in the model, or a line connecting two vertices in the model.

Panning

Pan [{Left|Right} <factor1>] [{Up|Down} <factor2>] [Screen | World] [Animation Steps <number_steps>]

Pan Cursor

Panning causes the camera to be moved up, down, left, or right. In terms of camera attributes, the **From** point and **At** point are translated equal distances and directions, while the perspective angle and up vector remain unchanged. The scene can also be panned by a factor of the graphics window size.

Screen and World indicate which coordinate system **<factor>** is in. If **Screen** is indicated (the default), **<factor>** is in screen coordinates, in which the width of the screen is one unit. If **World** is indicated, **<factor>** is expressed in the model units.

The **Pan Cursor** command is used to indicate the position of the desired view center with the mouse.

Zooming

Zoom Cursor [Click | Drag] [Animation Steps <number_steps>]

Zoom Screen <factor> [Animation Steps <number_steps>]

Zoom <x_min> <y_min> <x_max> <y_max> [Animation Steps <number_steps>]

Zoom {Group | Body | Volume | Surface | Curve | Vertex | Hex | Tet | Face | Tri | Edge | Node} <id_range> [Animation Steps <number_steps>] [Direction {options}]

Zoom Reset

After entering **Zoom Cursor**, move the cursor to the graphics window. If the **Click** option was entered, click on opposite corners of the desired zoom area; otherwise, drag a box around the area to zoom by holding down the left mouse button until the desired area is boxed in. **Click** is the default option for this command.

Zoom Screen will move the camera **<factor>** times closer to its focal point. The result is that objects on the focal plane will appear **<factor>** times larger.

Zooming on a specific portion of the screen is accomplished by specifying the zoom area in screen coordinates; for example, **Zoom 0 .25 .25** will zoom in on the bottom left quarter of the screen.

Zooming on a particular entity in the model is accomplished by specifying the entity type and ID after entering **Zoom**. The image will be adjusted to fit bounding box of the specified entity into the graphics window, and the specified entity will be highlighted. You can specify a final direction to look at when zooming by using the direction option.

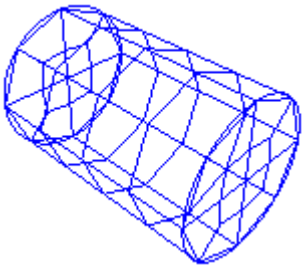
To center the view on all visible entities, use the **Zoom Reset** command.

Graphics Modes

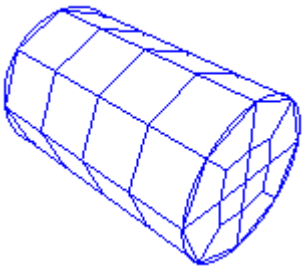
By default, the scene is viewed as a wireframe model. That is, only curves and edges are drawn, and surfaces are transparent. Surfaces can be drawn differently by changing the graphics mode:

> Graphics Mode {Wireframe | Hiddenline | Smoothshade | Transparent | Truehiddenline } [geometry | mesh]

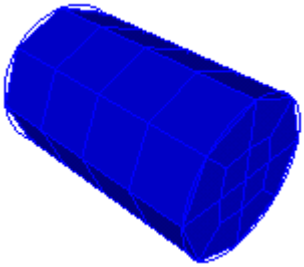
Examples and a brief description of each mode are shown below



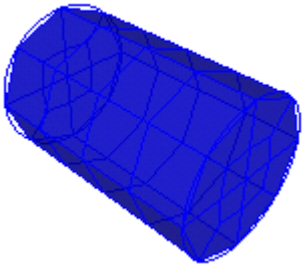
WireFrame - Surfaces are invisible. (This mode can also be accessed by typing **'wireframe'** at the command prompt.)



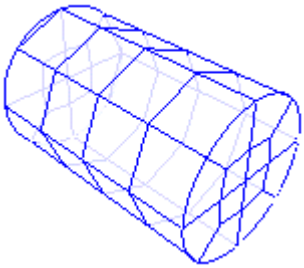
HiddenLine - Surfaces are not drawn, but they obscure what is behind them, giving a more realistic representation of the view. (This mode can also be accessed by typing **'hiddenline'** at the command prompt.)



SmoothShade - Surfaces are filled and shaded. Shaded colors are interpolated across the entire surface using the graphics lighting model. This produces the most realistic results. (This mode can also be accessed by typing **'shaded'** at the command prompt.)



Transparent - Renders surfaces as semi-transparent shaded images, allowing objects to shine-through from behind. Is not supported on all platforms, and generally requires advanced graphics hardware. (This mode can also be accessed by typing **'transparent'** at the command prompt.)



Truehiddenline - Similar to Hiddenline mode, but partly shows obscured lines. TrueHiddenLine mode also gives you additional options described below.

Truehiddenline Options

Graphics TrueHiddenLine Pattern <pattern>

This determines what pattern is used to draw lines behind surfaces (e.g. dotted, dashed, etc.; click here for a list of valid line patterns).

Displaying Using the Element Facets

There is another option that is similar to a graphics mode, set with the command

Graphics Use Facets [On|Off]

This command determines how shaded and filled surfaces are drawn when they are meshed. If Graphics Use Facets is on, the mesh facets (element faces) are used to render the model. This is particularly helpful for curved surfaces which may cut through some of the mesh faces. A comparison of graphics facets on and off is shown below.

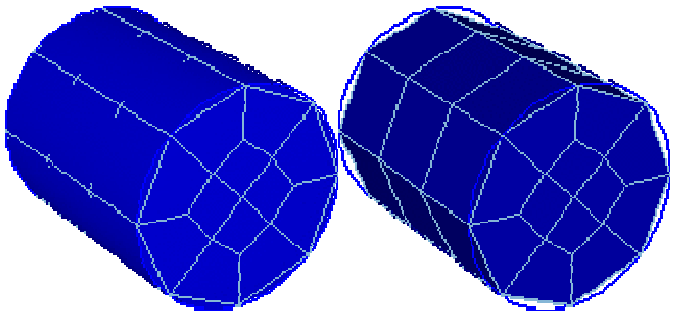


Figure 1. A meshed cylinder shown with graphics facets off (left) and graphics facets on (right); note how geometry facets on the curved surface obscure mesh edges when facets are off.

Drawing and Highlighting Entities

In order to effectively visualize the model, it is often necessary to draw an entity by itself, or several entities as a group. This is easily done with the command

Draw {Entity specification}[Add]

where Entity specification is an entity list as described in Command Line Entity Specification. This command clears the display before drawing the specified entity or entities. If the Add option is specified, the display is not cleared, and the given entity is added to what is already drawn on the screen. The entities specified in this command are drawn regardless of their visibility setting (see Geometry and Mesh Entity Visibility for more details about visibility).

Entities may also be drawn by selecting them with the mouse and then typing Ctrl-D while the mouse is in the graphics window. This will clear the screen and then draw only those entities that are currently selected.

Entities can be highlighted using the command

Highlight {Entity specification}

This command highlights the specified entities in the current display with the current highlight color. Highlighting can be removed using the command

Graphics Clear Highlight

To return to the normal display of the entire model, type Display.

Additionally, the visibility of individual entities, or sets of entities, can be controlled with the following visibility commands.

{Vertex|Curve|Surface|Volume|Body|Group} <range> [Geometry|Mesh] Visibility {on|off}

Edge [Visibility] {on|off}

{Mesh|Geometry} [Visibility]{on|off}

Drawing Other Objects

In addition to the common geometry, mesh and genesis entities, other objects may be drawn with variations of the Draw command. As with the other Draw commands, typing Display after drawing these objects will restore the scene to its normal display.

Displaying Entity Orientation

The normal to one or more surfaces, mesh faces, or mesh triangles may be drawn with the command

Draw {Surface | Face | Tri} <id_range> Normal [Length <length>] [Face | Tri]

The forward, or tangent, direction of a curve can be drawn with the command:

Draw Curve <id_range> Tangent [Length <length>]

If the Face or Tri qualifier is included in the Draw Normal command, the normals for all faces or tris that belong to the specified surface are drawn.

Volume Sources and Targets

Once the source and target surfaces have been set on a volume that will be meshed with the sweep algorithm, the source and target may be visually identified with the command

Draw Volume <volume_id_range> [Source][Target] [Length <size>]

If the Source keyword is included, the normal of the source surface or surfaces will be drawn in green into the specified volume. If the Target keyword is included, the normal of the target surface or surfaces will be drawn in red into the specified volume.

Model Axis

The model axis may be drawn with the command

Draw Axis [Length <length>]

The axis is drawn as three lines beginning at the model origin, one line in each of the three coordinate directions. The length of those lines is determined by the length parameter, which defaults to 1.

Surface Isoparameter Lines

Isoparameter lines may be drawn on surfaces in the model using the command

Draw Surface <surface_id_range> Isoparametric [Number <number>| [u <number>] [v <number>]]

If you specify the Number of lines, then the number of u- and v-parameter lines will be equal. You may specify instead a number of lines for each of the u and v parameters. The u-parameter lines will be drawn in red and the v-parameter lines will be drawn in blue.

Geometry Preview

Several options are available for previewing geometry without actually generating it. This is typically used in conjunction with webcutting and surface creation. The following Draw commands can be used for previewing geometry:

Draw Location On Curve

Draw Location

Draw Direction

Draw Axis

Draw Plane

Draw Cylinder

Mesh Slicing

A volume mesh can be viewed one layer at a time using a visualization tool known as mesh slicing. This tool divides the elements of one or more volumes into axis-aligned layers, and then allows the mesh to be displayed one layer at a time. Mesh slicing is especially useful to view the quality of swept meshes that are axis aligned.

Notes on Mesh Slicing

Mesh slicing is only intended to be a rough visualization tool. Because the average mesh edge length is used to determine the thickness of each layer, a layer may be more than one element deep. Unstructured meshes, meshes with large variations in edge length, and non-axis-aligned meshes will be more difficult to visualize with this tool.

The Graphical User Interface Version does not support the Mesh Slicing capability. Propagated Groups are an alternative to this capability.

Mesh Slicing Command

Mesh slicing can be started either by entering a keypress in the graphics window, which slices the mesh of the entire model, or by entering the command

Graphics Slice {Body | Volume} <id_range> Axis {X | Y | Z}

which slices only the bodies or volumes indicated, with a plane along the axis specified.

Key presses in the graphics window which control mesh slicing are summarized in the following table.

Key	Action
X,Y or Z	Initiate mesh slicing using the X, Y or Z plane
J	Move the slicing plane in the positive coordinate direction
K	Move the slicing plane in the negative coordinate direction
S	Toggles drawing single or multiple slice layers in the view
Q	Exit from mesh slicing mode

Entity Labels

Most entities may be labeled with text that is drawn at the centroid of the entity.

Mesh entities can be labeled with their ID number or their Genesis ID. Genesis ID labels are only valid after exporting a mesh.

Geometric entities can be labeled with their ID number or with other information.

Labels for groups of entity types can be turned on or off.

The following commands will accomplish this.

Label [on|off|name [only|id]|id|interval|size|merge|firmness]

Label All [on|off|name [only|id]|id|interval|size|merge|firmness]

Label Body [on|off| name [only|id] |id|interval|size| merge |firmness]

Label Curve [on|off|name [only|id] |id| interval| size| merge| firmness]

Label {Hex|Tet|Face|Tri|Edge} [on|off]

Label Geometry [on|off|name [only|id] |id| interval| size| merge| firmness]

Label Mesh [on|off]

Label Node [on|off|genesis]

Label Surface [on|off|name [only|id] |id| interval| scheme| size| merge| firmness]

Label Vertex [on|off|name [only|id] |id|interval| size| merge| firmness]

Label Volume [on|off|name [only|id] |id |interval| size |scheme |merge |firmness]

The meaning of each of each label type is listed below. Note that some label types don't make sense for every entity type.

On - The same as IDs.

Name - Name of the entity, if the entity has been named. Default name otherwise.

Name Only - If the entity has been named, use the name as the label. Otherwise, don't use a label.

Name IDs - If the entity has been named, use the name as the label. Otherwise, use the ID as the label.

Interval - The number of intervals set on the entity.

Firmness - Same as interval, but followed by a letter indicating the firmness of the interval setting (see the Mesh Generation chapter for description of firmness settings.)

Merge - Whether or not the entity is mergeable. Note that this is sometimes not clear, because, for example, a curve may show that it isn't mergeable because one of its owning surfaces may be unmergeable, while another owning surface may be mergeable.

Size - The mesh size set on this entity.

Note: Three dimensional entity types such as body will have their labels displayed in the center of the entity. Thus, in the **smooth shade** and **hidden line** graphics modes the labels will be hidden

Colors

Color Definitions

CUBIT has a palette of 85 pre-defined colors, listed in Appendix C. Users may also define their own colors in addition to those defined by CUBIT. Each color is defined by a name and by its RGB components, which range from 0 to 1.

To define an additional color, use either of the commands

Color Define "<name>" RGB <r g b>

Color Define "<name>" R <r> G <g> B <b>.

A maximum of 15 user-defined colors may be stored at one time, so it may be necessary to clear a color definition. This is done with the command

Color Release "<color_name>"

Color names can be listed with the command

Help Color

They are also listed in the appendix of this manual, along with their RGB definitions. To view a chart of color names and IDs, including those for user-defined colors, use the command

Draw Colortable

Specifying Colors in Commands

There are three ways to refer to a color in a command. They are

ID <id>

name

User "name"

The first of these three methods may be used for either pre-defined or user-defined colors. The second method is only valid for pre-defined colors, while the third is only valid for user-defined colors. Some examples of specifying colors in commands are:

By ID - Color Volume 1 ID 5

By Name (Pre-Defined) - Color Volume 1 Red

By Name (User-Defined) - Color Volume 1 User "mycolor"

Assigning Colors

Colors can be assigned to all geometric entities, and to some other objects as well. To assign a color to an entity or other object, use one of the following commands.

Color Axis Labels {<color_name>| id <color_id>}

Color Background {<color_name>| id <color_id>}

Color Block <block_id_range>{<color_name> | id <color_id>}

Color Body <body_id_range> [Geometry|Mesh] {<color_name>| id <color_id> | Default}

Color Curve <curve_id_range> [Geometry|Mesh] {<color_name>| id <color_id> | Default}

Color Group <group_id_range> [Geometry|Mesh] {<color_name>| id <color_id> | Default}

Color Highlight {<color_name>| id <color_id>}

Color Lines <color_name>

Color NodeSet <id_range> { <color_name> | id <color_id> | Default }

Color SideSet <id_range>{ <color_name> | id <color_id> | Default }

Color Surface <surface_id_range> [Geometry|Mesh] {<color_name>|Default}

Color Title {<color_name>|id <color_id>}

Color Volume <volume_id_range> [Geometry|Mesh] {<color_name>| id <color_id> | Default}

Color Whiskersheet <sideset_id_range> {<color_name> | id <color_id> | Default}

Including the Mesh keyword will change the color of the mesh belonging to the specified entity, without changing the color of the entity geometry itself. Conversely, including the Geometry keyword will change the geometry color without changing the mesh color. Including both keywords is identical to including neither keyword.

Colors are inherited by child entities. If you explicitly set the color for a volume, for example, all of its surfaces will also be drawn in that color. Once you assign a color to an entity, however, it will remain that color and will no longer follow color changes to parent entities. To make an entity follow the color of its parent after having explicitly set another color, use Default as the color name in the color command.

Colors can also be assigned to nodesets, sidesets, and element blocks. These colors do not take effect, however, unless the nodeset, sideset, or element block is drawn with a Draw command.

The background color and the color used to draw highlighted entities can be changed to any color.

By default, the axes are labeled with a white X, Y, and Z, indicating the three primary coordinate directions. If the background is changed to white, these labels are impossible to read; the color used to draw axis labels can be changed to any color. Changing the axis label color will change the text color for both the model axis and the triad (corner axis).

When several entity types are labeled, it can become difficult to determine which labels apply to which entities. To help distinguish which entities are being referred to by the labels, you may want to change the color of labels for specific entity types.

When a meshed surface is drawn in a shaded graphics mode, the mesh edges are not drawn in the same color as the surface. This is to prevent confusion between mesh edges and geometric curves, and to make the mesh edges more visible. The color used to draw mesh edges in this situation is known as the line color, and is gray by default; this color can be changed to any color.

Geometry and Mesh Entity Visibility

The visibility of geometric and mesh entities can be turned on or off, either individually, by entity type, by general entity class (mesh, geometry, etc.), or globally. Note that these commands do not refresh automatically. To refresh type **display** or **graphics flush** or click in the display window.

The commands to set the visibility are:

{ {Body|Curve|Surface|Volume} <range> } [Mesh][Geometry] Visibility [On|Off]

Edge Visibility [On | Off]

Vertex [Visibility] [on|off]

{Mesh|Geometry} { [Visibility] [on|off] }

If the **Mesh** keyword is included, only the visibility of the mesh belonging to the specified entity is affected. Similarly, if the **Geometry** keyword is included, only the visibility of the geometry is affected. Including neither keyword is identical to using both keywords.

Invisibility of geometry is inherited; visibility is not. For example, if a volume is invisible, its surfaces are also invisible unless they also belong to some other visible volume. As another case, if the volume is visible, but a surface is set to invisible, the surface will not follow its parent's visibility setting, but will remain invisible.

If **edge** visibility is off, mesh edges will not be drawn when mesh faces are drawn.

If **vertex** visibility is turned on, the vertices of the geometry become visible. The default for vertex visibility is off.

After turning mesh visibility off, all mesh will remain invisible until mesh visibility is turned on again. This is true no matter what other visibility commands are entered.

Similarly, after turning geometry visibility off, all geometry will remain invisible until geometry visibility is turned on again. This is true no matter what other visibility commands are entered.

Graphics Camera

One way to change what is visible in the graphics window is to manipulate the camera used to generate the scene. A scene camera has attributes described below, and depicted graphically in Figure 1. The values of these camera attributes determine how the scene appears in the graphics window.

Position (From) - The location of the camera in model coordinates.

View Direction (At) - The focal point of the camera in model coordinates.

Up Direction (Up) - The point indicating the direction to which the top of the camera is pointing. The Up point determines how the camera is rotated about its line of sight.

Projection - Determines how the three-dimensional model is mapped to the two-dimensional graphics window.

Perspective Angle - Twice the angle between the line of sight and the edge of the visible portion of the scene.

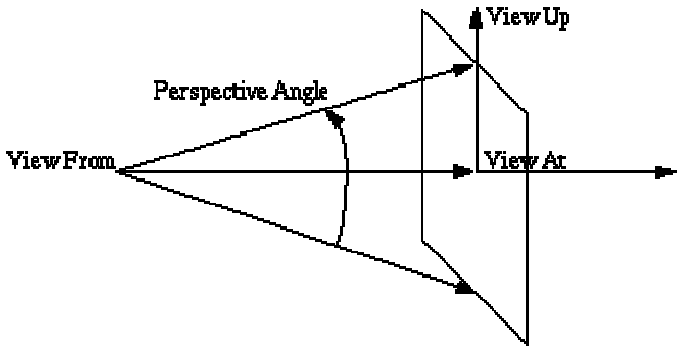


Figure 1: Schematic of From, At, Up, and Perspective Angle

At any time, the camera can be moved back to its original position and view using the command

View Reset

To see the current settings of these attributes, use the command

List View

The current value of the view attributes will be printed to the terminal window, along with other useful view information such as the current graphics mode and the width of the current scene in model coordinates.

Camera Attributes can be changed using the Rotate, Zoom and Pan commands, or directly as follows.

Changing Camera Attributes Directly

Camera attributes are most easily modified using interactive mouse manipulation (see Mouse-Based View Navigation) or using the rotate, pan and zoom commands. However, the camera attributes can also be modified directly with the following commands:

From <x y z>

At <x y z>

Up <x y z>

Graphics Perspective <On|Off>

Graphics Perspective Angle <degrees>

If **graphics perspective** is **on**, a perspective projection is used; if graphics perspective is off, an orthographic projection is used. With a perspective projection, the scene is drawn as it would look to a real camera. This gives a three-dimensional sense of depth, but causes most parallel lines to be drawn non-parallel to each other. If an orthographic projection is used, no sense of depth is given, but parallel lines are always drawn parallel to each other.

In a perspective view, changing the **perspective angle** changes the field of view by changing the angle from the line of sight to the edge of the visible scene. The effect is similar to a telephoto zoom with a camera. A smaller perspective angle results in a larger zoom. This command has no effect when graphics perspective is off.

Graphics Lighting Model

For shaded graphics display modes, the lighting model controls the intensity of the highlights and shadows for objects displayed in the graphics window. CUBIT offers two commands for controlling the lighting model.

Graphics Ambient Intensity {<intensity> | <r g b>}

Graphics Light Intensity {<intensity> | <r g b>}

The **ambient** intensity is the light available in the environment. There is no particular direction to the light source. In contrast, the **light** intensity is the effect of a simulated light source placed at the viewer's line of sight. The **light** intensity affects the intensity of the highlights and shadows, while the **ambient** intensity affects the brightness of the objects in the overall scene.

An **intensity** value from 0 to 1 can be used, where 0 represents no light and 1 represents maximum. Alternatively **r g b** color components can be used. This changes the color of the directional or ambient light source, affecting the resulting color of the objects in the model.

Graphics Window Size and Position

By default, CUBIT will create a single graphics window when it starts up (to run CUBIT without a graphics window, include -nographics on the command line when launching CUBIT.) The graphics window position and size is most easily adjusted using the mouse, like any other window on an X-windows screen. However, the size of the graphics window can also be controlled using the following commands:

Graphics WindowSize <width_in_pixels> <height_in_pixels>

Graphics WindowSize Maximum

Graphics WindowSize Minimum

After using the **Graphics WindowSize Maximum** and **Graphics WindowSize Minimum** commands, the previous window size can be restored by using the command

Graphics WindowSize Restore

In addition, on Unix workstations, the graphics window size and position can be controlled by placing the following line in the user's .Xdefaults file:

cubit.graphics.geometry XxY xpos ypos

where the **X** and **Y** are window width and height in pixels, respectively, and **xpos** and **ypos** are the offsets from the lower left hand corner.

Using Multiple Windows

You can use up to ten graphics windows simultaneously, each with its own camera and view. Each window has an ID, from 1 to 10, shown in the title bar of the window. Commands that control camera attributes apply to only one window at a time, the active window. Currently, the display lists of all windows are identical.

The following commands are used to create, delete, and make active additional graphics windows.

Graphics Window Create [ID]

Graphics Window Delete <ID>

Graphics Window Active <ID>

Saving Graphics Views

The current graphics view can be saved using the following command:

view save position <n>

When you save a view, you save the camera settings in effect at the time the command is issued. When you restore the view, the camera is returned to the saved position, orientation, and field of view.

If autocenter is on at the time you save the view, then restoring the view will automatically adjust the camera settings to center on the entire model and fit the entire model on the screen, a lot like "zoom reset." You turn autocenter on by typing "graphics autocenter on."

Example of how to **save a top view**:

at 0

from 0 1 0

up 1 0

graphics autocenter on

view save position 3

Use this command to restore that view:

view restore position 3

The view will then be looking down the y-axis, with the x-axis to the top and the z-axis to the right. The model will be centered in the view and zoomed so that everything just fits into the graphics window. This is true even if the model is not centered on the origin.

If autocenter is off when the "view save" command is issued, the camera is not adjusted to fit the scene into the graphics window. Instead, it is placed exactly where it was at the time the "save" command was issued.

Note that many graphics commands, such as "at", "from", and "up", do not change what appears in the graphics window until a "display" command is issued. They do, however, take immediate effect internally, and they do affect what is saved by the "view save" command.

In the command line version of CUBIT, you can save a view by holding down the control key and pressing one of the function keys (F1-F12). Each function key corresponds to a different saved view. A total of 12 views can be saved.. A view can be restored at a later time by pressing the appropriate function key WITHOUT holding down the control key.

It may be useful to save views in your cubit file so that they are available every time you run CUBIT. Use CUBIT to save front, top, and side views in positions 1, 2, and 3. If views are saved in your cubit file, it is convenient to add a "view reset" command after the views have been saved. Then the graphics will initially appear as they would if the view commands had not been included in your cubit file.

Hardcopy Output

CUBIT's Graphical User Interface provides the capability to print the contents of the graphics window directly to a printer.

In addition, a command line option is provided for dumping the contents of the graphics window to postscript or image files.

The command for generating hardcopy output files is:

Hardcopy '<filename>' {jpg | gif | bmp | pnm | tiff | eps} [window <window_id>]

Each of these options saves the view in the specified window (or the current window), to the specified file, in the format indicated. The file can then be sent to a printer or inserted into another document.

Screen Capture Programs

It should also be noted that many commercial applications are available for capturing screen images. In many cases, these applications may be more convenient for interactively capturing and saving a portion of the screen than the **Hardcopy** command discussed above. On UNIX platforms, the XV utility written by John Bradley is a good choice. In some cases this utility or its equivalent may be included with your system software. For Windows users, the *Print Screen* button will send a copy of the screen to the clipboard which can then be pasted into a paint program.

Miscellaneous Graphics Options

In addition to the commands discussed above, there are several other graphics system options in CUBIT that can be controlled by the user.

They include:

- Silhouette Lines
- Line Width
- Highlight Line Width
- Text Size
- Point Size
- Graphics Status
- Graphics Scale
- Model Axis
- Corner Axis
- Resetting the Graphics
- Shrink

Silhouette Lines

Some shapes, such as cylinders, are drawn with silhouette lines; these lines don't represent true geometric curves, but help visualize the shape of a surface. Silhouette lines can be turned on or off with the command

Graphics Silhouette [On|Off]

The pattern used to draw silhouette lines can be set using the command

Graphics Silhouette Pattern [solid | dashdot | dashed | dotted | dash_2dot | dash_3dot | long_dash | phantom]

LineWidth

This option controls the width of the lines used in the **wireframe**, **shaded**, **transparent**, **hiddenline** and **truehiddenline** displays. The default is 1 pixel wide. After using this command, it is necessary to refresh the graphics by either typing "display" or clicking the Refresh Graphics button. The command to set the line width is

Graphics LineWidth <width_in_pixels>

Highlight LineWidth

This option controls the width of the lines used when highlighting an entity. Setting this to a width greater than the global line width often makes it easier to locate highlighted entities. If this setting has not been changed, the line width set in the command above is used. After using this command, it is necessary to refresh the graphics by either typing "display" or clicking the Refresh Graphics button. The command to set the highlighting line width is

Highlight LineWidth <width_in_pixels>

Text Size

This option controls the size of text drawn in the graphics window. The size given in this command is the desired size relative to the default size. After using this command, it is necessary to refresh the graphics by either typing "display" or clicking the Refresh Graphics button. The command to set the text size is

Graphics Text Size <size>

Point Size

This option controls the size of points drawn in the graphics window, such as vertices or heads of vectors; alternatively, the size of points representing nodes or vertices can be set independently of the global point size. The commands to set the point sizes are

Graphics Point Size <size>

Graphics [Node | Vertex] Point Size <size>

Graphics Status

All graphics commands can be disabled or re-enabled with the command

Graphics {On | Off}

While graphics are off, changes in the model will not appear in the graphics window, and all graphics commands will be ignored. When graphics are again turned on, the scene will be updated to reflect the current state of the model.

Graphics Scale

A graphical scale can be drawn in the graphics window within the viewing area to obtain a bearing on model or part sizes. The command to turn the graphical scale on and off is:

Graphics Scale [On | Off]

Model Axis

The model axis may be drawn in the scene at the model origin. The axis is controlled with the command

Graphics Axis [Type <AXIS | Origin>] [on | off]

The command is used to specify whether the model axis is visible, and to determine how the axis is drawn. If you include Type Axis , the axis will be drawn as three orthogonal lines; if you include Type Origin, the axis will be drawn as a circle at the model origin.

Corner Axis (Triad)

By default, an axis appears in the corner of the graphics window. This corner axis, also called the triad, can be disabled or re-enabled with the command

Graphics Triad [On | Off]

Resetting the Graphics

Many of the graphic options can be reset back to default values with the command:

Graphics Reset

The graphic options set to defaults are:

- ambient and spot light intensity
- background color

- text size
- graphics mode
- silhouetting
- point size
- view type (Perspective)

In addition, this command also:

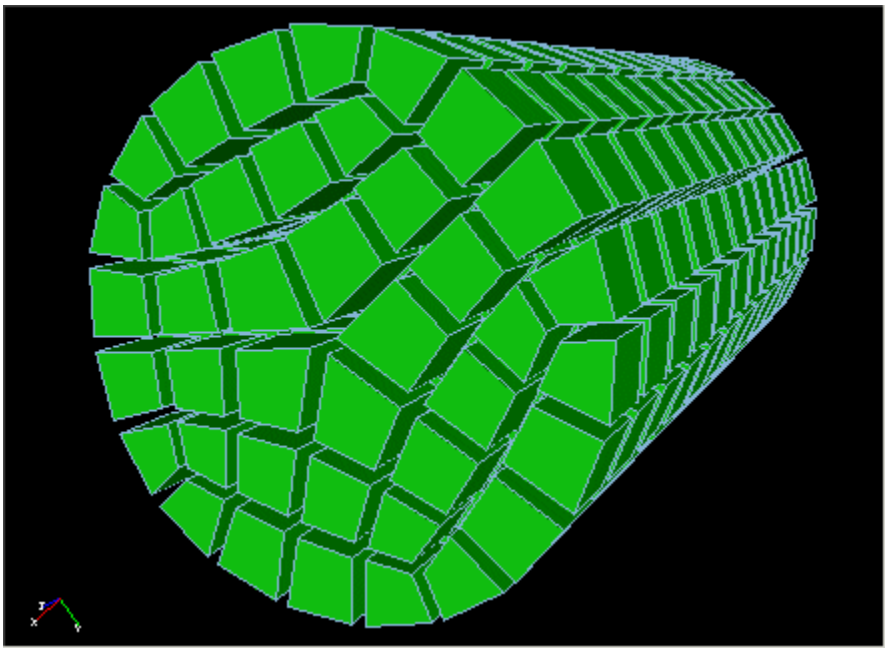
- centers the view on all visible entities (Zoom Reset)
- turns all labeling off
- turns vertex visibility off
- turns mesh and geometry visibility on
- moves the graphics camera back to its original position (View Reset)

Shrink

The shrink graphics attribute allows you to view the elements shrunken about their centroid. This is useful for viewing 3D meshes, permitting viewing of interior elements. It may also be useful for visually inspecting the mesh for missing elements. To use the shrink option use:

```
graphics shrink <value>
draw hex <range>
draw tet <range>
etc...
```

where **value** is a number between 0 and 1. One (1) will shrink the elements to a point, while zero (0) will not shrink the elements. The following figures illustrate the effect of element shrink on a hex mesh.



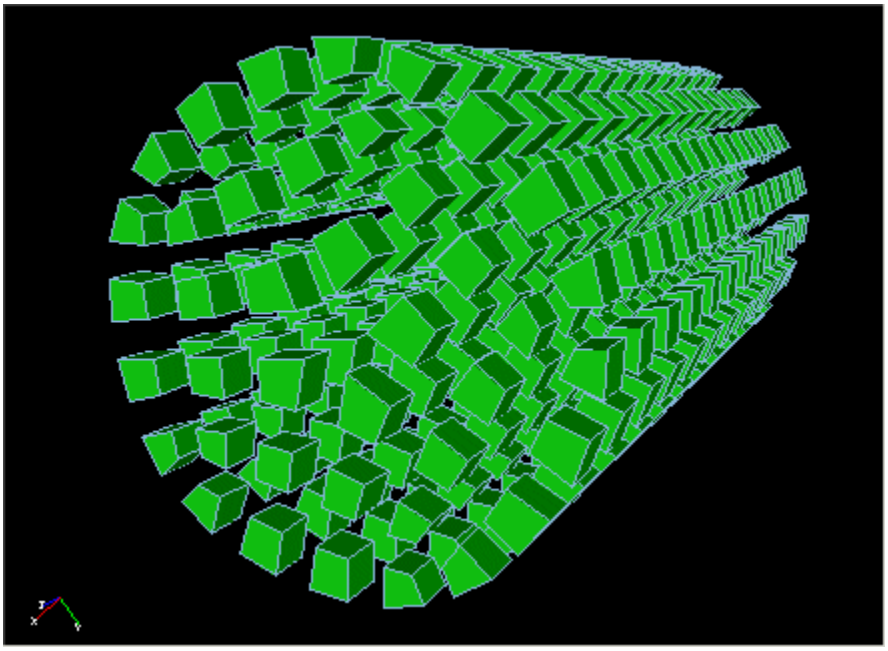


Figure 1. Top: shrink=0.2, Bottom: shrink=0.5

Entity Selection and Filtering

- Command Line Entity Specification
- Extended Command Line Entity Specification
- Selecting Entities With the Mouse
- Entity Filtering

CUBIT Entity specification is a means of selecting objects or groups of objects. Entities can be selected from the command line using entity filters, or directly in the graphics window using the mouse. This chapter describes these methods of entity selection.

Command Line Entity Specification

CUBIT identifies objects in the geometry, mesh, and elsewhere using ID numbers and sometimes names. IDs and names are used in most commands to specify which objects on which the command is to operate.

These objects can be specified in CUBIT commands in a variety of ways, which are best introduced with the following examples (the portion of each command which specifies a list of entities is shown in blue):

General ranges: Surface 1 2 4 to 6 by 2 3 4 5 Scheme Pave

Combined geometry, mesh, and genesis entities: Draw Sideset 1 Curve 3 Hex 2 4 6

Geometric topology traversal: Vertex in Volume 2 Size 0.3

Mesh topology traversal: Draw Edge in Hex 32

***All* keyword**: List Block all

***Expand* keyword**: my_curve_group expand Scheme Bias Factor 1.5

***Except* keyword**: List Curve 1 to 50 except 2 4 6

In addition to the examples above, there is an extended parsing capability that allows entities to be specified by a general set of criteria. This extended parsing capability is disabled by default in order to maintain journal file compatibility with earlier versions of CUBIT. In future versions of CUBIT (7.1 and above), extended parsing will be enabled by default. See Extended Entity Specification (data filters) for details. The following is a simple example of an extended entity specification:

By Criteria: Draw Curve With Length > 3

Types of Entity Range Input

The types of entity range input available in CUBIT can be classified in 4 groups:

1. General range parsing

Entity ids can be listed in ranges using multiple combinations of id lists (3 4 5 ...) and/or id ranges (1 to 7 by 2). In addition, the "all" identifier can be used anywhere a range is input. For example:

Draw Surface 1 2 4 to 6 Vertex all

2. Topological traversal

Topological traversal is indicated using the "in" identifier, can span multiple levels in a hierarchy, and can go either up or down the topology tree. For example, the following entity lists are all valid:

Vertex in Volume 3

Volume in Vertex 2 4 6

Curve 1 to 3 in Body 4 to 8 by 2

If ranges of entities are given on both sides of the "in" identifier, the intersection of the two sets results. For example, in the last command above, the curves that have ids of 1, 2 or 3 and are also in bodies 4, 6 and 8 are used in the command.

Topology traversal is also valid between entity types. Therefore, the following commands would also be valid:

Draw Node in Surface 3

Draw Surface in Edge 362

Draw Hex in Face in Surface 2

Draw Node in Hex in Face in Surface 2

Draw Edge in Node in Surface 2

3. Exclusion

Entity lists can be entered then filtered using the "except" identifier. This identifier and the ids following it apply only to the immediately preceding entity list, and are taken to be the same entity type. For example, the following entity lists are valid:

Curve all except 2 4 6

Curve 1 2 5 to 50 except 2 3 4

Curve all except 2 3 4 in surface 2 to 10

Curve in surface 3 except 2 (produces empty entity list!)

4. Group expansion

Groups in CUBIT can consist of any number of geometry entities, and the entities can be of different type (vertex, curve, etc.). Operations on groups can be classified as operations on the group itself or operations on all entities in the group. If a group identifier in a command is followed immediately by the `expand' qualifier, the contents of the group(s) are substituted in place of the group identifier(s); otherwise the command is interpreted as an operation on the group as a whole. If a group preceding the `expand' qualifier includes other groups, all groups are expanded in a recursive fashion.

For example, consider group 1, which consists of surfaces 1, 2 and curve 1. Surfaces 1 and 2 are bounded by curves 2, 3, 4 and 5. The commands in Table 1, illustrate the behavior of the `expand' qualifier.

Extended Command Line Entity Specification

In addition to basic entity specification, entities may be specified using an extended expression. An extended expression identifies one or more entities using a set of entity criteria. These criteria describe properties of the entities one wishes to operate upon.

Extended Parsing Syntax

All expressions that are valid when extended parsing is disabled are also valid when extended parsing is enabled. In other words, you may still specify a list of entities by ID range, topological relationship, etc.The most common type of extended expression is in this format:

{Entity_Type} With {Criteria}

Entity_Type is the name of any type of entity that can be used in a command, such as Curve, Hex, or SideSet. Criteria is a combination of entity properties (such as Length), operators (such as >=), keywords (such as Not), and values (such as 5.3) that can be evaluated to true or false for a given entity. Here are some examples:

Curve With Length <1 Surface With Is_Meshed = FalseNode With X_Coord > 10 And Y_Coord > 0

Keywords

These are the keyword defined by extended parsing **All, To, Step, By, Except, In, Expand**

These keywords are used the same way as in basic entity specification. For example:

Draw Surface All Draw Surface 1 To 5 Step 2 Curve 1 to 3 in Body 4 to 8 by 2 Draw Hex in Face in Surface 2 Draw Node in Hex in Face in Surface 2 Curve 1 2 5 to 50 except 2 3 4

Not - Not flips the logical sense of an expression - it changes true to false and false to true. For example:

Draw Surface With Not Is_Meshed

Of - The of operator is used to get an attribute value for a single entity, such as "length of curve 5". Only attributes that return a single numeric value may be used in an "of" expression. There must be only one entity specified after the "of" operator, but it can be identified using any valid entity expression. An example of a complete command which includes the "of" operator is:

List Curve With Length < Length Of Curve 5 ids

Operators <, >, <=, >=, =, <> - These relational operators compare two expressions. You may use = or == for "equals". <> means "not equal". For example:

Draw Surface With X_Max <= 3Draw Volume With Z_Max <>12.3

Arithmetic Operators +, -, *, / These arithmetic operators work in the traditional manner.

Draw Surface With Length * 3 + 1.2 > 10

And, Or - These logic operators determine how multiple criteria are combined.

Draw Surface With Length > 3 Or With Is_Meshed = False

( ) - Parentheses are used to group expressions and to override precedence. When in doubt about precedence, use parentheses.

Draw Surface With Length > 3 And (With Is_Meshed = False Or X_Min > 1)

Functions

The following functions are defined. Not all functions apply to all entities. If a function does not apply to a given entity, the function returns 0 or false.

ID - The ID of an entity.

Length - The length of a curve or edge.

Exterior_Angle - Works for curves with an exterior angle greater than (>), less_than (<), or equal to (=) a given angle in degrees. This is used if you want to do some operation, such as refinement, on all the reentrant curves or curves with surfaces that form a certain angle.

Is_Meshed - Whether a geometric entity has been meshed or not.

Is_Spline - Whether a geometric entity is defined using a NURBS representation. Otherwise the entity has an analytic representation.

Element_Count -The number of elements owned by this geometric entity. Only elements of the same dimension as the entity are counted (number of hexes in a volume, number of faces on a surface, etc.).

Dimension -The topological dimension of an entity (3 for volumes, 2 for surfaces, etc.).

X_Coord, Y_Coord, Z_Coord - The x, y, or z coordinate of the point at the center of the entity's bounding box.

X_Min, Y_Min, Z_Min -The x, y, or z coordinate of the minimum extent of the entity's bounding box.

X_Max, Y_Max, Z_Max - The x, y, or z coordinate of the maximum extent of the entity's bounding box.

Is_Merged - Whether a geometry entity has a merge flag on. All geometric entities have one set by default.

Is_Virtual - A flag that specifies whether an entity is virtual geometry. An entity is virtual if it has at least one virtual (partition/composite) topology bridge.

Has_Virtual - An entity "has_virtual" if it is virtual itself, or has at least one child virtual entity

Is_Real - An entity "is_real" if it has at least one real (non-virtual) topology bridge.

Precedence

For complicated expressions, which entities are referred to is influenced by the order in which portions of the expression are evaluated. This order is determined by precedence. Operators with high precedence are evaluated before operators with low precedence. You may always include parentheses to determine which sub-expressions are evaluated first.Here all operators and keywords listed from high to low precedence. Items listed together have the same precedence and are evaluated from left to right.

(,) Expand Not *, / +, - <, >, <=, >=, <>, = And, Or Except In Of With

Because of precedence, the following two expressions are identical:

Curve With Length + 2 * 2 > 10 And Length <= 20 In my_group

Expand(Curve With (((Length + (2*2)) > 10)And(Length <= 20))) In (my_group Expand)

Selecting Entities with the Mouse

The following discussion is applicable only to the command line version of CUBIT. See GUI Entity Selection for a description of interactive entity selection with the Graphical User Interface.

Many of the commands in CUBIT require the specification of an entity on which the command operates. These entities are usually specified using an object type and ID (see Entity Specification) or a name. The ID of a particular entity can be found by turning labels on in the graphics and redisplaying; however, this can be cumbersome for complicated models. CUBIT provides the capability to select with the mouse individual geometry or mesh entities. After being selected, the ID of the entity is reported and the entity is highlighted in the scene. After selecting the entities, other actions can be performed on the selection. The various options for selecting entities in CUBIT are described below, and are summarized in Table 1:

Table 1. Picking and key press operations on the picked entities

Key	Action
ctrl + B1	Pick entity of the current picking type.
shift + ctrl + B1	Add picked entity of the current picking type to current picked entity list.
tab	Query-pick; pick entity of current picking type that is below the last-picked entity.
n	Lists what entities are currently selected.
l	Lists basic information about each selected entity. This is similar to entering a List command for each selected entity.
g	Lists geometric information about the selection. As if the List Geometry command were issued for each entity. If there are multiple entities selected, a geometric summary of all selected entities is printed at the end, including information such as the total bounding box of the selection.
i	Makes the current selection invisible. This only affects entities that can be made invisible from the command line (i.e. geometric entities.)
s	Draws a graphical scale showing model size in the three coordinate axes. This is a toggle action, so pressing the 's' key again in the graphics window will turn the scale off.
ctrl + z	Zoom in on the current selection.
e	Echo the ID of the selection to the command line.
a	Add the current selection to the picked group. Only geometry will be added to the group (not mesh entities). If a selected entity is already in the picked group, it will not be added a second time.
r	Remove the current selection from the picked group. If a selected entity was not found in the picked group, this command will have no effect.
ctrl + r	Redisplays the model.
c	Clear the picked group. The picked group will be empty after this command.
m	Lists what entities are currently in the picked group.

d	Display and select the entities in the picked group.
ctrl + d	Draws the entity that is selected.

Details of selecting entities with a mouse are outlined in the following items:

- Entity Selection
- Query Selection
- Multiple Selected Entities
- Information about the Selection
- Picked Group
- Substituting the Selection into Commands

Entity Selection

Selecting entities typically involves two steps:

1. Specifying the type of entity to select

Clicking on the scene can be interpreted in more than one way. For example, clicking on a curve could be intended to select the curve or a mesh edge owned by that curve. The type of entity the user intends to select is called the picking type. In order for CUBIT to correctly interpret mouse clicks, the picking type must be indicated. This can be done in one of two ways. The easiest way to change the picking type is to place the pointer in the graphics window and enter the dimension of the desired picking type and an optional modifier key. The dimension usually corresponds to the dimension of the objects being picked:

Table 2. Picking Modes in Graphics Window

Number	Default pick	Number +shift pick
0	vertices	nodes
1	curves	edges
2	surfaces	all 2D elements
3	volumes	all 3D elements
4	bodies	

If a Shift modifier key is held while typing the dimension, the picking type is set to the mesh entity of corresponding dimension, otherwise the geometry entity of that dimension is set as the picking type. For example, typing 2 while the pointer is in the graphics window sets the picking type so that geometric surfaces are picked; typing Shift-1 sets the picking type so that mesh edges are picked. To differentiate between picking "tris" or "quads" use "**pick face**" or "**pick tri**"

The picking type can also be set using the command

Pick <entity_type>

where entity_type is one of the following: Body , Volume , Surface , Curve , Vertex , Hex , Tet , Face , Tri , Edge , Node , or DicerSheet .

2. **Selecting the entities**

To select an object, hold down the control key and click on the entity (this command can be mapped to a different button and modifiers, as described in the section on Mouse-Based View Navigation). Clicking on an entity in this manner will first de-select any previously selected entities, and will then select the entity of the correct type closest to the point clicked. The new selection will be highlighted and its name will be printed in the command window.

Query Selection

If the highlighted entity is not the object you intended to selected, press the Tab key to move to the next closest entity. You can continue to press tab to loop through all possible selections that are reasonably close to the point where you clicked. Shift-Tab will loop backwards through the same entities.

Multiple Selected Entities

To select an additional entity, without first clearing the current selection, hold down the shift and control keys while clicking on an object. You can select as many objects as you would like. By changing the picking type between selections, more than one type of entity may be selected at a time. When picking multiple entities, each pick action acts as a toggle; if the entity is already picked, it is "unpicked", or taken out of the picked entities list.

Information About the Selection

When an entity is selected, its name, entity type, and ID are printed in the command window. There are several other actions which can then be performed on the picked entity list. These actions are initiated by pressing a key while the pointer is in the graphics window. Table 1 summarizes the actions which operate on the selected entities.

Picked Group

There is a special group whose contents can be altered using picking. This group is named picked , and is automatically created by CUBIT. Other than its relationship to interactive picking, it is identical to other groups and can be operated on from the command line. Like other groups, both geometric and mesh entities can be held in the picked group. Table 1 lists the graphics window key presses used with the picked group.

Note: It is important to distinguish between the current selection and the picked group contents. Clicking on a new entity will select that entity, but will not add it to the picked group. De-selecting an entity will not remove an entity from the picked group.

Substituting Selection into Other Commands

There are three ways to use mouse-based selection to specify entities in commands.

1. The Selection Keyword

You may refer to all currently selected entities by using the word selection in a command; the picked type and ID numbers of all selected entities will be substituted directly for selection . For example, if Volume 1 and Curve 5 are currently selected, typing

Color selection Blue

is identical to typing

Color Volume 1 Curve 5 Blue

Note that the selection keyword is case sensitive, and must be entered as all lowercase letters.

2. Echoing the ID of the Selection

Typing an e into a graphics window will cause the ID of each selected entity to be added to the command line at the current insertion point. This is a convenient way to use entities of which you don't already know the name or ID.

As an added convenience, the picking type can be set based on the last word on the command line using the ` key. Note that this is not the apostrophe key, but rather the left tick mark, usually found at the upper-left corner of the keyboard on the same key as the tilde (~). For example, a convenient way to set the meshing scheme of a cylinder to sweep would be as follows:

Volume (hit `, select cylinder, hit e) Scheme Sweep Source Surface (hit `, select endcap, hit e) Target (select other endcap, hit e)

The result will be something similar to

Volume 1 Scheme Sweep Source Surface 1 Target 2

Notice that you must use the word Surface in the command, or ` will not select the correct picking type.

3. Using the Picked Group in Commands

Like other groups, the picked group may be used in commands by referring to it by name. The name of the picked group is picked. For example, if the contents of the picked group are Volume 1 and Volume 2, the command

Draw picked

is identical to

Draw Volume 1 Volume 2

Note that picked is case sensitive, and must be entered as all lowercase letters.

Entity Filtering

CUBIT entity filtering provides the user with the capability to quickly select the entities needed and to parse out those that match certain entity characteristics. In general, think of filtering as starting with a list of entities, then passing that list through a filter or set of filters which removes entities not meeting the filter criteria. In most cases, filters are defined using the Filter Dialog when the Graphical User Interface is installed. The following is a description of their command line implementation.

In general there are two modes to filtering:

1. Execute a filter immediately. This allows you to list, draw, highlight or select entities that meet a certain attribute criteria. For example, draw all the surfaces with scheme pave or draw all elements with a certain quality characteristic.
2. Register a filter which is used in further operations. Register a filter which is used in further operations. Registered filters can be used to filter entities from the mouse pick list or keyed-in entity lists. Filters can be chained together (in a Boolean AND/OR mode) to result in entities meeting a certain set of criteria.

The following describes the Execute and Register Filter features in CUBIT:

Executing Filters

Quality Filter Commands

Execute Filter Commands

Registering Filters

Filtering with Registered Filters

Filter Operations

Register Filter Commands

Executing Filters

There are a large variety of filter types that can be executed immediately. It is generally recommended that the filters be run from the GUI, where you can see all of the types available. An example of the syntax is:

Execute [filter] {Volume|Surface|Curve} Mesh_Scheme <mesh_scheme> [INCLUDE | Exclude] [on {Volume | Surface | Curve} <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

The default is to run on all entities of the specified type; optionally you can give an entity list (i.e., get all paved surfaces in volumes 1 to 3). If you use the **INCLUDE** default, the filter will return all entities matching the input. If the **EXCLUDE** option is used, all entities other than the input will be returned (i.e., all surfaces not paved). The **Execute** command will always result in a listing of the entities - you can also draw or highlight them in the graphics. When drawing, if you use the **Add** qualifier they will be added to the display; otherwise the graphics are cleared first then they are drawn. The **Select** option will select the entities in the GUI (i.e., they will be current on the right mouse button and copied into the current selection list, if it matches the type of entities being filtered). You can also place the filtered entities into a group.

Quality Filter Commands

The filters support quality ranges of hexes or faces. Here is an example that draws the 10 hexes with the worst jacobian values in the model:

Execute filter hex quality_range Jacobian bottom 10 include on hex all draw

This command will draw all the unshared element edges in the model, useful for finding cracks in a mesh:

Execute filter edge owned_hexes equal_to 1 include on edge all draw

Other similar filters exist.

Execute Filter Commands

Following is a list of currently supported execute filter commands:

Group Type

Execute [filter] Group Type <group_type> [INCLUDE | exclude] [on Group <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Is Meshed

Execute [filter] {Group | Body | Volume | Surface | Curve | Vertex} Is_Meshed [INCLUDE|exclude] [on {Group | Body | Volume | Surface | Curve | Vertex}] <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Is Visible

Execute [filter] {Group | Body | Volume | Surface | Curve | Vertex} Is_Visible [INCLUDE|exclude] [on {Group | Body | Volume | Surface | Curve | Vertex}] <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Group Ref Entities Only

Execute [filter] Group Ref_Entities_Only [INCLUDE | exclude] [on Group <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Group Mesh Entities Only

Execute [filter] Group Mesh_Entities_Only [INCLUDE | exclude] [on Group <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Group Is Empty

Execute [filter] Group Is_Empty [INCLUDE | exclude] [on Group <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Body Volume

Execute [filter] Body Volume {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [INCLUDE|exclude] [on Volume <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Interval Number

Execute [filter] {Body | Volume | Surface | Curve} Interval_Number {greater_than|less_than|equal_to <value>} [INCLUDE | exclude] [on {Body|Volume|Surface|Curve} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Body Is Sheet

Execute [filter] Body Is_Sheet [INCLUDE|exclude] [on Body <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Body Contained Volumes Equal To

Execute [filter] Body Contained_Volumes Equal_To <number> [INCLUDE | exclude] [on Body <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Contained Surfaces Equal To

Execute [filter] {Body|Volume} Contained_Surfaces Equal_To <number> [INCLUDE|exclude] [on {Body|Volume} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Contained Curves Equal To

Execute [filter] {Body | Volume | Surface} Contained_Curves Equal_To <number> [INCLUDE | exclude] [on {Body | Volume | Surfaces} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Contained Vertices Equal To

Execute [filter] {Body | Volume | Surface | Curve} Contained_Vertices Equal_To <number> [INCLUDE | exclude] [on {Body | Volume | Surface | Curve} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Volume Range

Execute [filter] Volume Range {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [INCLUDE | exclude] [on Volume <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Mesh Scheme

Execute [filter] {Volume | Surface | Curve} Mesh_Scheme <mesh_scheme> [INCLUDE | exclude] [on {Volume | Surface | Curve} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Smooth Scheme

Execute [filter] {Volume | Surface} Smooth_Scheme <mesh_scheme> [INCLUDE|exclude] [on {Volume|Surface} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Surface Area

Execute [filter] Surface Area {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [INCLUDE | exclude] [on Surface <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Surface Is Souce

Execute [filter] Surface Is_Source [INCLUDE | exclude] [on Surface <id_range>] [Draw|Highlight] [into group {id | 'name'}] [select] [add]

Surface Is Target

Execute [filter] Surface Is_Target [INCLUDE | exclude] [on Surface <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Surface Is Linking

Execute [filter] Surface Is_Linking [INCLUDE|exclude] [on Surface <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Is Periodic

Execute [filter] {Surface | Curve} Is_Periodic [INCLUDE | exclude] [on {Surface | Curve} <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Attached Volumes Equal To

Execute [filter] Surface Attached_Volumes Equal_To <number> [INCLUDE | exclude] [on Surface <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Merge Partners

Execute [filter] {Surface | Curve | Vertex} Merge_Partners [Equal_To | Less_Than | Greater_Than} <number>] [INCLUDE | exclude] [on {Surface | Curve | Vertex} <id_range>] [Draw | Highlight] [into group {<id> | 'name'}] [select] [add]

Quality Range

Execute [filter] {Hex|Face} Quality_Range <metric_name> {Low <double> |High <double>| Top <integer>| Bottom <integer>} [INCLUDE | exclude] [on Hex | Face <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Geometry Type

Execute [filter] {Surface|Curve} Geometry_Type <geometry_type> [INCLUDE | exclude] [on {Surface | Curve} <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Sizing Function

Execute [filter] Surface Sizing_Function <sizing_function_type> [INCLUDE|exclude] [on Surface <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Length

Execute [filter] {Curve | Edge} Length {greater_than | greater_than_equal | less_than|equal_to | less_than_equal <value>} [INCLUDE | exclude] [on {Curve | Edge} <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Attached Surfaces Equal To

Execute [filter] Curve Attached_Surfaces Equal_To <number> [INCLUDE | exclude] [on Curve <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Curve Interval Setting

Execute [filter] Curve Interval_Setting {Equal_To | Less_Than | Greater_Than <number>} [INCLUDE | exclude] [on Curve <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Curve Is Tolerant

Execute [filter] Curve Is_Tolerant [INCLUDE | exclude] [on Curve <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Attached Curves Equal To

Execute [filter] Vertex Attached_Curves Equal_To <number> [INCLUDE | exclude] [on Vertex <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Color

Execute [filter] {Body | Volume | Surface | Curve | Vertex | Hex | Face | Edge | Node} Color <type> [INCLUDE|exclude] [on {Body | Volume | Surface | Curve | Vertex | Hex | Face | Edge | Node} <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Related To

Execute [filter] {Group | Body | Volume | Surface | Curve | Vertex | Hex | Face | Edge | Node} Related_To [on {Body | Volume | Node | Edge | Face} <id_range>] [INCLUDE | exclude] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Full Hex

Execute [filter] Hex Full_Hex [INCLUDE | exclude] [on Hex <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Node Hex

Execute [filter] Hex Node_Hex [INCLUDE | exclude] [on Hex <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Owned Hexes Equal To

Execute [filter] {Face | Edge} Owned_Hexes Equal_To <number> [INCLUDE | exclude] [on {Face|Edge} <id_range>] [Draw|Highlight] [into group {id|'name'}] [select] [add]

Free Faces

Execute [filter] Face Free_Faces [INCLUDE | exclude] [on Face <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Owned Faces Equal To

Execute [filter] Edge Owned_Faces Equal_To <number> [INCLUDE | exclude] [on Edge <id_range>] [Draw | Highlight] [into group {id|'name'}] [select] [add]

Free Edges

Execute [filter] Edge Free Edge [INCLUDE | exclude] [on Edge <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add]

Node position Fixed

Execute [filter] Node Position_Fixed [INCLUDE | exclude] [on Node <id_range>] [Draw | Highlight] [into group {id | 'name'}] [select] [add**]**

Registering Filters

When filters are registered they can be used in conjunction with mouse picking and entity parsing. By default, the entities found during a mouse pick list will be passed through any registered filters. If any entities are removed during this process a warning message will be echoed to the output window. You can control whether filtering occurs during mouse selection with this command:

set Pick Filter [ON|off]

In GUI dialogue entity list fields, the registered filters will always filter the input list. On the command line you can cause draw, highlight and list commands to use the filters with the following setting:

set Parse Filter [on|OFF]

The draw, highlight and list commands can be override this setting by the options [**filter | no filter]**.

There currently is not a way to filter every input list on the command line nor turn off the registered filters in GUI dialogue input lists.

Some of the GUI pages register filters internally - this can help the user to avoid unnecessary picks. For example, when webcutting with a plane normal to a curve, the end of the curve needs to be picked. Thus, after the curve is selected a filter is registered allowing only vertices attached to the end of the selected curve to be picked. When picking you will be reminded that certain items are being filtered out via an information message in the output window. This can be very helpful when trying to pick vertices that are coincident with other vertices, etc.. This is called a *tool defined* filter. User registered filters are called *user defined*.

An example of a command to register a filter follows:

> Register [filter] Surface Geometry_Type {spline | plane | cone | cylinder | sphere | torus | best fit] [and | or] [include | exclude]

Filtering with Registered Filters

After the filters are registered, they can be exercised with:

> Filter <entity> <id range>

For example, if a surface filter was registered (i.e. register surface related_to volume 1), the command:

> Filter surface all

will display the ids of the surfaces that pass this filter. If there are more than one surface filter is registered, the filter command will consider all of them.

If an entity is to be filtered with respect to other registered entity filters, the range identifier "in" can be used to consider those other entity filters. For example, suppose there are node and face filters registered. To filter hex entities with consideration to the face and node filters, the command would be:

> Filter hex in face <id range> in node <id range>
>
> *{filter hex in face all in node all}*

The effect of this command will be to first parse and filter the node entities; then to parse and filter the face entities with respect to the nodes that pass the node filter(s). Then to parse and filter (if there are hex filters) the hex entities with respect to the faces that pass the previous face filter(s).

Filter Operations

List

The filter(s) registered can be listed by entity or they can all be listed at one time. To list by entity, the command is:

> List <entity_type> Filter *{i.e. list curve filter}*

To list all the filters, the command is: **list filter all**

Suppose these commands have been entered,

> register curve length equal_to 5 and register curve mesh_scheme bias

An example of the **list curve filter** command output is shown below:

```
________filter type________ _______status_______
1.) curve_length ( = 5)      {include}    active  User Defined
2.) curve_mesh_scheme (bias) {or, include}active  User Defined
```

Suppress/Resume

Each registered filter is at first "active". The user can make this filter inactive by suppressing the filter. To suppress the filter, the command is:

Filter Suppress <entity> Filter <number>

Using the example above, to suppress the curve_mesh_scheme filter, the command is:

filter suppress curve filter 2

To make it active again, the command is:

filter resume curve filter 2

All the filters can be suppressed/ unsuppressed at one time with the command:

Filter {suppress | resume} All

Move

If there are multiple filters, these filters can be re-arranged. The command is:

Filter Move <entity> Filter <number> to <number>

Using the curve filters example above, to move the curve_mesh_scheme above the curve_length filter, the command is:

filter move curve filter 2 to 1

The resulting **list curve filter** command would now output:

```
_____filter type_______    _______status_______
1.) curve_mesh_scheme(bias) {include}    active User Defined
2.) curve_length ( = 5)     {or,include} active User Defined
```

Delete

Once a filter has been registered, it can be deleted with the command:

Filter Delete <entity_type> Filter <number>

Using the curve filter examples again, to delete the first filter (curve_length), the command is:

filter delete curve filter 1

All the filters can be deleted at one time with the command:

filter delete all

Register Filter Commands

A list of the currently available register commands follows:

Curve Related To

Register [filter] Curve Related_To {body | volume | surface | vertex <id>} [and | or] [include | exclude]

Curve Geometry Type

Register [filter] Curve Geometry_Type {arc | segmented | spline | line | point}

Curve Length

Register [filter] Curve Length {greater_than | greater_than_equal|less_than | equal_to | less_than_equal <value>} [and | or] [include | exclude]

Curve Attached Surfaces Equal To

Register [filter] Curve Attached_Surfaces Equal_To <number> [and | or] [include | exclude]

Curve Mesh Scheme

Register [filter] Curve Mesh_Scheme {bias | dualbias | equal | featuresize | morph} [and | or] [include | exclude]

Curve Is Tolerant

Register [filter] Curve Is_Tolerant [and | or] [include | exclude]

Vertex Related To

Register [filter] Vertex Related_To {curve | surface | volume | body <id>} [and | or] [include | exclude]

Attached Curves Related To

Register [filter] Vertex Attached_Curves Equal_To <number> [and | or] [include | exclude]

Surface Area

Register [filter] Surface Area {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [and | or] [include | exclude]

Attached Volume Equal To

Register [filter] Surface Attached_Volumes Equal_To <number> [and | or] [include | exclude]

Surface Mesh Scheme

Register [filter] Surface Mesh_Scheme {map | pave | parallelpave | pentagon | submap | triangle | trimap | trimesh | tripave} [and | or] [include | exclude]

Smooth Scheme

Register [filter] {surface | volume} Smooth_Scheme <type> [and | or] [include | exclude]

possible smooth schemes are: laplacian (free/fixed), equipotential (free/fixed), isoparametric, centroid area pull (free/fixed), replicate, optimize (free/fixed/jacobian/optms) winslow (free/fixed), randomize

Surface Sizing Function

Register [filter] Surface Sizing_Function {constant | linear | curvature | super | test | exodus| inv_int | interval | none} [and | or] [include | exclude]

Surface Geometry Type

Register [filter] Surface Geometry_Type {spline | plane | cone | cylinder | sphere | torus | best fit] [and | or] [include | exclude]

Surface Sweep Type

Register [filter] Surface {is_linking | is_source | is_target} [and | or] [include | exclude]

Surface Related To

Register [filter] Surface Related_To {vertex | curve | volume | body <id>} [and | or] [include | exclude]

Volume Range

Register [filter] Volume Range {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [and | or] [include | exclude]

Volume Mesh Scheme

Register [filter] Volume Mesh_Scheme {dice | map | plaster | pyramid | sweep } [and | or] [include | exclude]

Volume Related To

Register [filter] Volume Related_To {vertex | curve | surface | body} [and | or] [include | exclude]

Body Volume

Register [filter] Body Volume {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [and | or] [include | exclude]

Body Related To

Register [filter] Body Related_To [vertex | curve | surface | volume<id>] [and | or] [include | exclude]

Body Is Sheet

Register [filter] Body Is_Sheet [and | or] [include | exclude]

Body Contained Volume Equal To

Register [filter] Body Contained_Volumes Equal_To <number> [and | or] [include | exclude]

Merge Partners

Register [filter] {curve | surface | vertex} Merge_Partners [equal_to | greater_than | less_than <number>] [and | or] [include | exclude]

Is Meshed

Register [filter] {vertex | curve | surface | volume | body} Is_Meshed [and | or] [include | exclude]

Is Periodic

Register [filter] {curve | surface} Is_Periodic [and | or] [include | exclude]

Color

Register [filter] {vertex | curve | surface | volume | body | hex | face | edge | node} Color <color> [and | or] [include | exclude]

Visibility

Register [filter] {vertex | curve | surface | volume | body} Visibility [and | or] [include | exclude]

Interval Number

Register [filter] {curve | surface | volume | body} Interval_Number {equal_to | greater_than | less_than <number>} [and | or] [include | exclude]

Interval Setting

Register [filter] {curve | surface | volume | body} Interval_Setting {default | soft | hard} [and | or] [include | exclude]

Smooth Scheme

Register [filter] {surface | volume} Smooth_Scheme <type> [and | or] [include | exclude]

Possible smooth schemes are: laplacian (free/fixed), equipotential (free/fixed), isoparametric, centroid area pull (free/fixed), replicate, optimize (free/fixed/jacobian/optms) winslow (free/fixed), randomize

Contained Surface Equal To

Register [filter] {body | volume} Contained_Surfaces Equal_To <number> [and | or] [include | exclude]

Contained Curves Equal To

Register [filter] {body | volume | surface} Contained_Curves Equal_To <number> [and | or] [include | exclude]

Quality Range

Register [filter] {hex | face} Quality_Range <metric_name> {low | high | top | bottom <number> | malformed} [and | or] [include | exclude]

Possible metric names are: aspect ratio, aspect ratio gamma, skew, taper, warpage, element area, stretch, maximum/minimum angle, oddy, folding, jacobian, element volume diagonal ratio, dimension, scaled jacobian

Hex Related To

Register [filter] Hex Related_To {body | volume | node | face | edge <id>} [and | or] [include | exclude]

Hex FullHex

Register [filter] Hex Full_Hex [and | or] [include | exclude]

Hex NodeHex

Register [filter] Hex Node_Hex [and | or] [include | exclude]

Face Related To

Register [filter] Face Related_To {body | volume | surface | hex | edge | node<id>} [and | or] [include | exclude]

Edge Related To

Register [filter] Edge Related_To {body | volume | surface | curve | vertex | hex | face | node<id>} [and | or] [include | exclude]

Node Related To

Register [filter] Node Related_To {body | volume | surface | curve | vertex <id>| hex | face | edge} [and | or] [include | exclude]

Node Position Fixed

Register [filter] Node Position_Fixed [and | or] [include | exclude]

Edge Length

Register [filter] Edge Length {greater_than | greater_than_equal | less_than | equal_to | less_than_equal <value>} [and | or] [include | exclude]

Location, Direction and Axis Specification

- Specifying a Location
- Specifying a Location on a Curve
- Specifying a Direction
- Specifying an Axis

Many commands require that a location or a direction be specified. Although entering the three floating point numbers required to uniquely define a vector is perfectly acceptable, it may be more convenient to specify the direction or location with respect to existing entities in the model.

For example, the following commands might be used for creating straight curves using location and direction specification described here:

Create Curve [From] Location {options} Location {options}

Create Curve [From] Location {options} Direction {options} Length <val>

Specifying a Location

Some commands require a specified location or point (such as create curve spline) for the command. A location is basically an x-y-z position in the model. The following options determine a location specification:

- **[Position] <xval yval zval>**
- **Last**
- **[At] {Node|Vertex} <id>**
- **[On] Curve <range> [location on curve options]**
- **[On] Surface <id> [Location {options} | CENTER]**
- **Center Curve <id>**
- **Extrema {Curve|Surface|Volume|Body|Group} <range> [Direction] {options} [Direction {options}]**
- **Between { Location <options> Location <options>} | { Location <options> Project {Curve|Surface} <id> } [Stop] [Fraction <val>] }**
- **[Move [all] {<xval yval zval> | {Dx|X|Dy|Y|Dz|Z} <val> | Direction {options} Distance <val>}]**
- **[Swing [all] [About] Axis {options} Angle <ang>]**
- **Multiple Location Specification**

Position (XYZ values)

[Position] <xval yval zval>

The most basic way to specify a location is to just give the xyz values of the location. In this case the following two commands both draw a location at the coordinates (1, 2, 3), as the Position keyword is optional:

draw location position 1 2 3
draw location 1 2 3

Last Location Used in a Command

Last

The last option recalls the last location used in a command. For example, if the following command is entered after the above position commands a location would be drawn at the position (1, 2, 3).

draw location last

Last locations do not carry over from CUBIT session to CUBIT session. The last location defaults to (0, 0, 0) if no location has been used during the session.

Node or Vertex

[At] {Node|Vertex} <range>

Referring to a node or vertex simply returns the coordinates of that node or vertex. The following draws a location at the coordinates of Vertex 5:

draw location vertex 5

On a Curve

Various options are available to specify a location on a curve. See the section Specifying a Location On a Curve for details.

On a Surface

[On] Surface <range> [Location {options} | CENTER]

If a surface is used to specify a location without other options, the geometrical center of the surface is found (the center keyword is optional - the default). Otherwise, you can specify another general location and that location is projected to the surface. For example, the following command will draw the location that is position (5,0,0) projected to surface 1:

draw location on surface 1 location 5 0 0

Any valid location options can be used to specify the location that is projected to the surface.

Center

Center Curve <range>

Finds the center of an arc - an error is returned if the curve is not an arc.

Extrema

Extrema {Curve|Surface|Volume|Body|Group} <range> [Direction] {options} [Direction {options}] [Direction {options}]

The extrema option returns the location of the maximum value, on the specified entity or group, in the specified direction. For example, the following places a vertex on a surface at the point of maximum y-axis value.

create vertex location extrema surf 1 direction y

Between

Between {Location <options> Location <options> } | {Location <options> Project {Curve|Surface} <range>} [Stop] [Fraction <val>]}

The between option finds a location that is between two locations or a location and an entity. An optional fraction can be given to specify the fractional distance from the first location to the second location or entity. For example, the following will draw a location at (5, 0, 0):

draw location between location 0 0 0 location 10 0 0

The following will draw a location at (2.5, 0, 0) - 25% of the distance from (0, 0, 0) to (10, 0, 0):

draw location between location 0 0 0 location 10 0 0 fraction .25

The second item can be an entity:

draw location between location 0 0 0 vertex 2
draw location between location 0 0 0 surface 1

In the second case, location (0, 0, 0) is projected to surface 1, then the location that is between (0, 0, 0) and the projected location is found.

Of course, any valid location can be used in the command. In the following example a location at the top center of the brick is found:

brick x 10
draw location between location bet vert 3 vert 2
location bet vert 8 vert 5

The first location is between vertices 3 and 2, and the second location is between vertices 8 and 5.

Move

Move {<xval yval zval> | {Dx|X|Dy|Y|Dz|Z} <val> |
Direction {options} Distance <val>

Any location can be optionally moved either a xyz distance or a certain distance in a given direction. As many moves as desired can be strung together. For example, the following will return a location at (5, 0, 0):

draw location 0 0 0 move 5 0 0

These examples add another move that basically moves the location (5, 0, 0) in a direction 45 degrees up and to the right a distance of 10 (all three commands are equivalent - see sections on directions and rotations):

draw location 0 0 0 move 5 0 0 move {10*sind(45)} {10*sind(45)} 0
draw location 0 0 0 move 5 0 0 move direction 1 1 0 distance 10
draw location 0 0 0 move 5 0 0 move direction 1 0 0 rotate about 0 0 1 angle 45 dist 10

Swing

Swing [About] Axis {options} Angle <ang>

Any location can be "swung" (rotated) about an axis by a certain angle. (See the section on specifying an axis for the axis syntax). As with moves, multiple swings can be strung together. The following example rotates the location (2.5, 5, 5) thirty degrees about an axis defined by Curve 11. Note that the right-hand rule is used to determine the direction of the swing about the axis.

draw location 2.5 5 5 swing about axis curve 11 angle 30

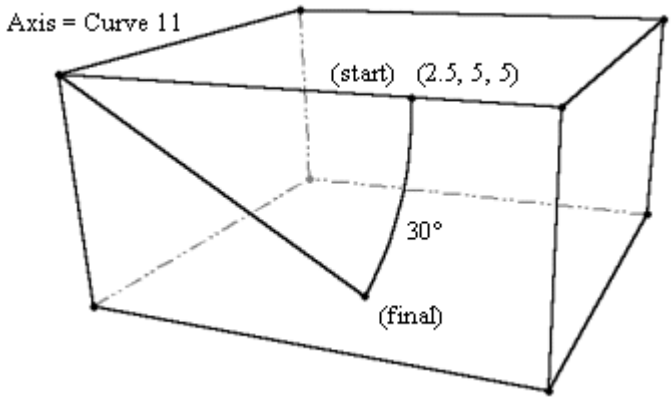


Figure 1 - Swinging a Location

Multiple Location Specification

Location {options} Location {options}...

Multiple location specifications can be used in a single command. For example, the following command uses several locations to create a spline curve at points (0,0,0), (1,2,3), (4,5,6), and (7,8,9).

create curve spline location 0 0 0 location 1 2 3 location 4 5 6 location 7 8 9

Previewing a Location

Sometimes it is advantageous to preview a location before using it in a command. A location can be previewed with the Draw command. All of the options that can be used to specify locations in a command can be used to preview locations as well. See above for a description of these options. The command syntax is:

Draw Location {options}

Specifying a Location on a Curve or Curves

Some commands require you to specify a location on a curve (i.e., webcutting with a plane normal to a curve). The following are the options for specifying a location (or locations in the case of the segment option) on a curve:

- **{ MIDPOINT | Start | End | }**
- **Fraction <val 0.0 to 1.0> [From Vertex <id> | Start|End]**
- **Distance <val> [From {Vertex|Curve|Surface} <id> | start | end]**
- **{{Close_To|At} Location {options} | Position <xval><yval><zval> | {Node|Vertex} <id>}**
- **Extrema [Direction] {options} [Direction {options}] [Direction {options}]**
- **Segment <num_segs>**
- **Crossing {Curve|Surface} <id_list> [Bounded|Near] }**
- **Previewing a Location**

Start, Midpoint, or End

{ MIDPOINT | Start | End |

These options simply specify the location that is the midpoint, start or end point of a curve. By default, the midpoint is the understood location unless another location is specified.

Fraction

Fraction <val 0.0 to 1.0> [From Vertex <id> | Start|End] |

The fraction option simply finds the location that is a fractional distance along the curve. By default, the fraction references the start of the curve; however, you can optionally specify which vertex to reference from.

Distance

Distance <d> [From {Vertex|Curve|Surface} <id> | start | end] |

The distance option not only can find a location that is a certain distance along the curve from the start or end of the curve, but can also find a location (or locations if there is more than one solution) on a curve that is a specified distance from another curve or a surface. If the From Curve option is used both curves must lie in the same plane.

draw location on curve 13 distance 7 from curve 2

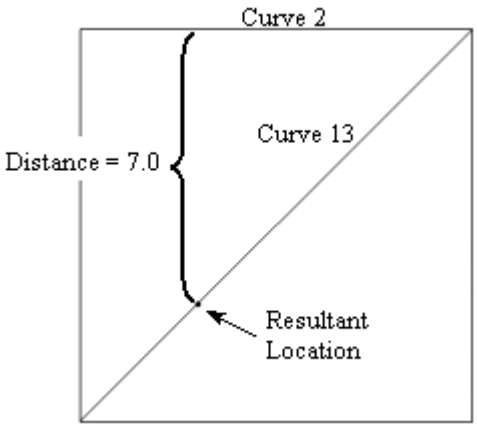


Figure 1 - Location on a Curve a Distance from Another Curve

{Close_To|At} Location

{{Close_To|At} Location {options} | Position <xval><yval><zval> |{Node|Vertex} <id>} |

These options take a location closest to the location on the curve.

Extrema

Extrema [Direction] {options} [Direction {options}] [Direction {options}]

The extrema option finds the maximum value location along a curve in a specified direction. For example:

create vertex location on curve 1 extrema ny

Creates a vertex on curve 1 at the location where the y axis value of the curve is at a minimum.

Segment

Segment <num_segs>

The segment option finds locations spaced evenly along the curve such as to break the curve into equal length "segments" (of course the curve is not modified). You must specify a minimum of two segments (if two segments were specified a location would be found at the center of the curve). The following example results in 4 locations:

draw location on curve 1 segment 5

create vertex on curve 1 segment 5

Figure 2 - Five Segments on a Curve

Crossing

Crossing {Curve|Surface} <id_list> [Bounded|Near] }

The crossing option finds locations at the intersection of the curve and another curve or surface. By default, the curve(s) and surface are extended to infinity and the intersections are calculated; if the bounded option is specified only intersections that lie on the bounded entities will be returned. The near option is valid only for two linear curves. If near is specified the nearest location between the two linear curves will be returned.

Previewing a Location on a Curve

A location on a curve can be previewed with the Draw command. All of the options that can be used for specifying a location on a curve can be used to preview a location on a curve. See above for a description of these options. The command syntax is:

Draw Location On Curve <curve id> {options}

Specifying a Direction

Some commands require a specified a direction or vector for the command. A direction is basically a xyz vector in the model. The following options determine a direction specification:

- **[Vector] <xval yval zval>**
- **Last**
- x|y|z|nx|ny|nz
- **[On] | [Tangent] [At] Curve <id> {location on curve options}**
- [On] | [Normal] [At] Surface <id> [Location {options}]
- **[From] { Location {options} | {Node|Vertex} <id> }[Project] {Location {options} | [Entity] {Node|Vertex|Curve|Surface} <id> }**
- **[Rotate {options}]**
- **[Cross [With] Direction {options}]**
- **[Reverse]**

Vector (XYZ values)

[Vector] <xval yval zval>

The most basic way to specify a direction is to just give the vector x-y-z components of the direction. The given vector need not be a unit vector. The following three commands simply draw a direction in the x-direction (1, 0, 0) as the Vector keyword is optional and unit vectors are not required:

```
draw direction vector 1 0 0
draw direction 1 0 0
draw direction 10 0 0
```

Last Direction Used

Last

The last option recalls the last direction used in a command. For example, if the following command is entered after the above vector commands a direction location would be drawn in the x-direction (1, 0, 0).

```
draw direction last
```

Last directions do not carry over from CUBIT session to CUBIT session. The last direction defaults to (1, 0, 0) if no direction has been used during the session.

X|Y|Z|Nx|Ny|Nz

x|y|z|nx|ny|nz

The x|y|z|nx|ny|nz options assign the x direction, y direction, z direction, negative x direction, negative y direction and negative z direction respectively.

On Curve Tangent

[On] | [Tangent] [At] Curve <id> {location on curve options}

The curve option simply finds a tangent vector on a curve. Note that the on, tangent and at keywords are optional, as well as the location on the curve. If no location is specified, the tangent at the start vertex of the curve is found. See the section above, Specifying a Location on a Curve, for details on how to specify where along the curve the tangent vector is found.

```
draw direction curve 1
draw direction on curve 1
draw direction tangent at curve 1
draw direction tangent at curve 1 distance 3
draw direction tangent at curve 1 fraction .5
draw direction tangent at curve 1 distance 2 reverse
```

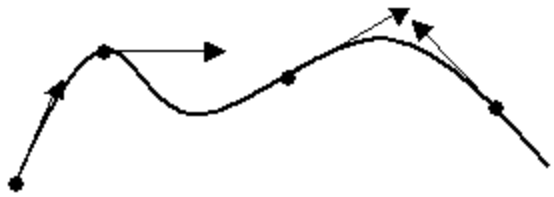


Figure 1 - Tangents to a Curve

On Surface Normal

[On] | [Normal] [At] Surface <id> [Location {options}]

The surface option simply finds a normal vector on a surface. Note that the "on", "normal" and "at" keywords are optional, as well as the location on the surface. If no location is specified, the normal vector at the center of the surface is found. If a location is specified, the location is projected to the surface, then the normal vector is found.

draw direction on surface 1
draw direction on surface 1 location 1 2 0

From Location

[From] {Location {options} | Node|Vertex <id>} [Project] {Location {options} | [Entity] {Node|Vertex|Curve|Surface} <id>}

The from location option finds a direction that is from one location to another or from a location to an entity. If the second specification is an entity, the first location is projected to the entity to find the direction.

draw direction from vertex 1 vertex 2
draw direction from location on curve 1 fraction .5 surface 3

Note that when using an entity for the second specification, the Project and Entity keywords are generally optional. However, it is sometimes necessary to remove ambiguity from the previous location specification. For example, the following will not parse correctly:

draw direction location on curve 1 distance 2 surface 3

In this case, the location on the curve is parsed as a distance 2.0 from surface 3. Instead, the desired behavior is to find the location on curve 1 as a distance of 2.0 along the curve from the start of the curve, and project it to surface 3 to find the direction. The following commands (all equivalent) achieve this behavior:

draw direction location on curve 1 distance 2 project surface 3
draw direction location on curve 1 distance 2 entity surface 3
draw direction location on curve 1 distance 2 project entity surface 3

Rotate

[Rotate {options}]

The rotate option allows you to rotate the direction about another vector. You can string together as many rotations as necessary. For example:

draw direction 1 0 0 rotate about z 135 rotate about curve 1 angle 50

Options that can be used with rotate are as follows:

{Ax|X|Ay|Y|Az|Z [Angle] <angle>} | { {[About] | Towards} Direction {options} Angle <val> } [Rotate (options)] [Origin (location)]

Ax, Ay, Az (or X,Y,Z) angles can be entered in any order. The optional specification of another rotate keyword in the options indicated that multiple nested rotations are permitted.

Cross

[Cross [With] Direction {options}]

The cross option allows you to find the vector cross product of the direction with another direction.

Reverse

[Reverse]

This keyword simply reverses the direction specification.

Previewing a Direction

Sometimes it is helpful to preview a direction before using it in a command. A direction may be previewed using the Draw command. The direction options are described above. See Specifying a Location for a list of location options.

Draw Direction {direction_options} [Location (location_options)]

Specifying an Axis

Some commands require a specified axis (such as webcut with a cylinder) and it is sometimes advantageous to view an axis before modifying geometry. An axis is simply a vector with a specified origin. The following options determine an axis specification:

- Last
- Specify a direction and a location
- Revolve an axis about an axis

Last

Last

The last option recalls the last axis used in an axis command. The last axis does not carry over from CUBIT session to CUBIT session.

Specify an origin and a vector

{Direction {options} [Origin [Location] {options}] [Length <val>] [Angle <val>]}

To specify an axis simply specify a vector (a direction) and an origin (a location). Notice that the command requires the axis direction first because the origin defaults to 0 0 0 when not specified. An example of specifying an axis to draw a location using the swing command is as follows:

draw location 1 0 0 swing about axis direction z ang 45

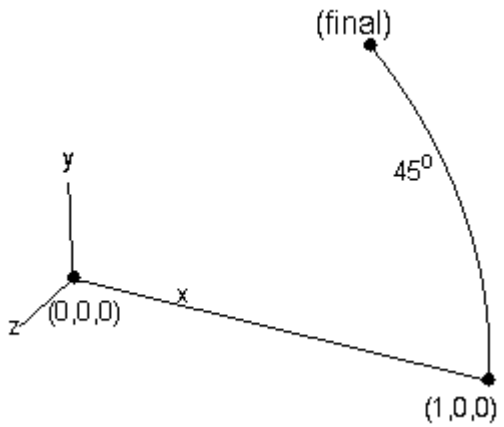


Figure 1 - Swinging a point about the z-axis

The location 1 0 0 was swung 45 degrees about an axis defined by a vector in the z direction and an origin at 0 0 0.

Revolve an axis about an axis

[Axis {options} Revolve [About] Axis {options} Angle <val>]

To revolve one axis around another use the revolve keyword. The following example revolves the first axis (defined by the y-axis and origin) around the second axis (defined by the z-axis and origin) by 45 degrees and draws the result.

draw axis direction y revolve axis direction z angle 45

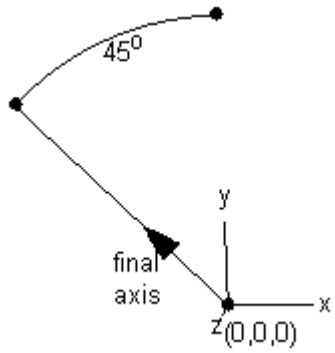


Figure 2 - Revolving an axis about another axis

Previewing an Axis

Sometimes it is helpful to preview an axis before using it in a command. An axis may be previewed using the Draw command. The options for previewing an axis are the same as the ones described above.

Draw Axis {options}

Drawing a Location, Direction, or Axis

Some commands require you to specify a location on a curve (i.e., webcutting with a plane normal to a curve). This location can be previewed with the following options:

1. A fraction along the curve from the start of the curve, or optionally, from a specified vertex on the curve.
2. A distance along the curve from the start of the curve, or optionally, from a specified vertex on the curve.
3. An xyz position that is moved to the closest point on the given curve.
4. The position of a vertex that is moved to the closest point on the given curve.

Draw Location On Curve <curve id> {Fraction <f> | Distance <d> | Position <xval><yval><zval> | Close_To Vertex <vertex_id>} [[From] Vertex <vertex_id> (optional for 'Fraction' & 'Distance')]

Some commands require a specified axis (such as webcut with a cylinder) and it is sometimes advantageous to view an axis before modifying geometry. To draw a preview of an axis use the following command:

Draw Axis {options}

Some commands require a specified location or point (such as create curve spline) and it is sometimes advantages to view a location before modifying or creating geometry. To draw a preview of a location use the following command:

Draw Location {options}

Listing Information

The **List** commands print information about the current model and session. There are five general areas: *Model Summary, Geometry, Mesh, Special Entities,* and *CUBIT Environment*. The descriptions of these areas includes example output based on the model generated by a journal file listed below. The model consists of a 1x2x3 brick meshed with element size 0.1.

- List Model Summary
- List Geometry
- List Mesh
- List Special Entities
- List CUBIT Environment

Journal File Used for List Examples

```
brick x 1 y 2 z 3
body 1 size 0.1
mesh volume 1
block 1 volume 1
nodeset 1 surface 1
sideset 1 surface 2
group "my_surfaces" add surface 1 to 3
surface 2 name "BackSurface"
surface 3 name "BottomSurface"
surface 1 name "FrontSurface"
surface 4 name "LeftSurface"
surface 5 name "RightSurface"
surface 6 name "TopSurface"
```

List Model Summary

The following commands print identical summaries of the model: the number of entities of each geometric, mesh, and special type

List Model

List Totals

The following output is generated from the **list model** command.

CUBIT> list model

```
Model Entity Totals:
    Geometric Entities:
        0 assemblies
        0 parts
        2 groups
        1 bodies
        1 volumes
        6 surfaces
       12 curves
        8 vertices
    Mesh Entities:
      6000 hexes
        0 pyramids
        0 tets
      7876 faces
        0 tris
      9854 edges
      7161 nodes
    Special Entities:
        1 element blocks
        1 sideSets
        1 nodesets
```

Journaled Command: list model

List Geometry

The following commands list information about the geometry of the model.

list names [group|body|volume|surface|curve|vertex|all]

list {group | body | volume | surface | curve | vertex} <range> [ids]

list {geom_list} [Geometry|Mesh [Detail]]

list {group | body | volume | surface | curve | vertex} <range> {x|y|z}

The first command lists the names in use, and the entity type and id corresponding to each name. Specifying **all** lists names for all types; other options list names for a specific entity type. The names for an individual entity can be obtained by listing just that entity. Sample output from the list names surface command is shown below. This output shows that, for example, Surface 2 has the name ` BackSurface '.

```
______Name_____   __Type__    Id      _Propagated_
    BackSurface    Surface     2          No
  BottomSurface    Surface     3          No
   FrontSurface    Surface     1          No
    LeftSurface    Surface     4          No
   RightSurface    Surface     5          No
     TopSurface    Surface     6          No
```

List Names Example

The second command provides information on the number of entities in the model and their identification numbers. If a range is given then detailed information is given on each entity in that range, unless the **ids** option is also given. If the **ids** option is used, just a list of ids is printed. This list can be very useful for large models in which several geometry decomposition operations have performed. Sample output from the list surface command is shown below.

CUBIT> list surface ids
The 6 surface ids are 1 to 6.

CUBIT> list surf ids
The 108 surface ids are 192 to 266, 268 to 271, 273 to 301.

List Surface [range] Ids' Examples

The **<range>** can be very general using the general entity parsing syntax. Using a **<range>** gives a brief synopsis of the local connectivity of the model, e.g. one can list the ids of the surfaces containing vertex 2; as shown in the listing below.. An intermediately detailed synopsis can be obtained by placing the range of entities in a group, then listing the group.

```
CUBIT> list surface in vertex 2 ids
The 3 entity ids are 1, 5, 6.

CUBIT> group "v2_surfs" equals surface in vertex 2
CUBIT> list v2_surfs Group Entity 'v2_surfs' (Id = 3)
It owns/encloses 3 entities: 3 surfaces.
Owned Entities:            Mesh Scheme   Interval:   Edge
_____Name____   Type______Id +is meshed    Count       Size
FrontSurface   Surface   1        map+      1 H       0.1
  TopSurface   Surface   6        map+      1 H       0.1
RightSurface   Surface   5        map+      1 H       0.1
```

Using 'List' for Querying Connectivity.

The third command provides detailed information for each of the specific entities. This information includes the entity's name and id, its meshing scheme and how that scheme was selected, whether it is meshed and other meshing parameters such as smooth scheme, interval size and count. The entity's connectivity is summarized by a table of the entity's subentities and a list of the entity's superentities. Also, the nodesets, sidesets, blocks, and groups containing the entity are listed.

Specifying **geometry** will additionally list the extent of the entity's geometric bounding box, the geometric size of the entity, and depending on entity type, other information such as surface normal. See also the **list {entities} x** command below.

Specifying **mesh** will additionally list the number of mesh entities of each type interior to the entity and on bounding subentities. **Mesh detail** will list the ids of the mesh entities as well, following the format of the **list ids** command above.

The fourth command lists the entities sorted by either the x, y, or z coordinate of their geometric center. For example, in a large, basically cylindrical model centered around z-axis, it is useful to list the surfaces of a volume sorted by z to identify the source and target sweeping surfaces.

List Mesh

The following commands list mesh entity information.

list { hex | face | edge | node } <id_range>

list { hex | face | edge | node } <id_range> ids

For both of these commands, the range can be very general, following the general entity parsing syntax. The first command provides detailed information. For an entity, the information includes its id, owning geometry, subentities and superentities. For a hex, the Exodus Id is also listed. For a node, its coordinates are listed. The second command just lists the entity ids, and is usually used in conjunction with complex ranges.

List Special Entities

List {special_type} [range]

Special entities include (element) blocks, sidesets and nodesets (representing boundary conditions). Like the **list geometry** and **list mesh** commands, if no range is specified then the number of entities of the given type is summarized. Otherwise, listing a special entity prints the mesh and geometry it contains.

(Some special entities are of interest mainly to developers and are not described here, e.g. whisker sheets, whisker hexes, and dicer sheets.)

List Cubit Environment

The user may list information about the current CUBIT environment such as message output settings, memory usage, and graphics settings.

Message Output Settings

There are several major categories of CUBIT messages.

- **Info** (Information) messages tell the user about normal events, such as the id of a newly created body, or the completion of a meshing algorithm.
- **Warning** messages signal unusual events that are potential problems.
- **Error** messages signal either user error, such as syntax errors, or the failure of some operation, such as the failure to mesh a surface.
- **Echo** messages tell the user what was journaled.
- **Debug** messages tell developers about algorithm progress. There are many types of Debug messages, each one concentrating on a different aspect of CUBIT.

By default, Info, Warning, Error, and Echo messages are printed, and Debug messages are not printed. Information, Warning and Debug message printing can be turned on or off (or toggled) with a set command; error messages are always printed. Debugging output can be redirected to a file. Current message printing settings can be listed.

List {echo | info | errors | warning | debug }

Set {echo | info | warning } [on|off]

[Set] Debug <index> [on|off]

[Set] Debug <index> File <'filename'>

[Set] Debug <index> Terminal

Message flags can also be set using command line options, e.g. -**warning={on|off}** and -**information={on|off}.** Debug flags can be set on with **-debug=<setting>,** where **<setting>** is a comma-separated list of integers or ranges of integers denoting which flags to turn on. E.g. to set debug flags 1, 3, and 8 to 10 on, the syntax is -**debug=1,3,8-10**.

In addition to the major categories, there are some special purpose output settings.

[Set] Logging {off|on file <'filename'> [resume]}

List Logging

If logging is enabled, all echo, info, warning, and error messages will
be output both to the terminal and to the logging file. The **resume** option will append to the logfile, if it exists, instead of writing over it. If the logfile doesn't already exist, it will be created.

List Journal
Title "<title_string>"

The List Journal command lists which types of CUBIT commands will be journaled and the file to which the journaled commands are being written.

List Title

The List Title command will list the title to be written to the Exodus file. To assign a title to an exodus file, use the Title command.

List Default Block
Set Default Block {ON|off}

The List Default Block command lists which type of geometric entities for which blocks will automatically be generated at export if no other blocks have been specified. The Set Default Block command will toggle whether these default blocks are written, or not, during the export operation when no other blocks have been specified.

List Settings

The List Settings command lists the value of all the message flags, journal file and echo settings, as well as additional information. The first section lists a short description of each debug flag and its current setting. Next come the other message settings, followed by some flags affecting algorithm behavior.

Sample output

```
CUBIT> list settings

Debug Flag Settings (flag number, setting, output to, description):

 1  OFF  terminal           Debug Graphics toggle for some debug options.

 2  OFF  terminal           Whisker weaving information

 3  OFF  terminal           Timing information for 3D Meshing routines.

 4  OFF  terminal           Graphics Debugging (DrawingTool)

 5  OFF  terminal           FastQ debugging

 6  OFF  terminal           Submapping graphics debugging

 7  OFF  terminal           Knife progress whisker weaving information

 8  OFF  terminal           Mapping Face debug / Linear Programming debug

 9  OFF  terminal           Paver Debugging

.

.

.

echo               = On

info               = On

journal            = On

journal graphics   = Off

journal names      = On

journal aprepro    = On

journal file       = 'cubit11.jou'

warning            = On

logging            = Off

recording          = Off

keep invalid mesh = Off

default names      = Off

default block     = Volumes

catch interrupt    = On

name replacement character = '_', suffix character = '@'
```

```
Matching Intervals is fast, TRUE;

multiple curves will be fixed per iteration.

Note in rare cases 'slow', FALSE, may produce better meshes.

Match Intervals rounding is FALSE;

intervals will be rounded towards the user-specified intervals.
```

Graphical Display Information

List View

List view prints the current graphics view and mode parameters; See Graphics Window .

Memory Usage Information

Users are encouraged to use Unix commands such as `top' to check total CUBIT memory use. Developers may check internal memory usage with the following command:

List Memory [`<object type>']

Without an object type, the command prints memory use for all types of objects.

Measuring Distances Between Entities

To output various properties of entities, the following **Measure** command options are available.

- Measure Between
- Measure Small

Measure Between

Measure Between {Vertex|Curve|Surface |Volume|Node} <id | id_list>
[{Vertex|Curve|Surface|Volume|Node} <id>]

The **Measure Between** command outputs the distance from one entity to the next. The result will always be the minimum distance between entities. For example, measuring between two spheres will output the minimum distance between them, not the distance between centroids. The example shown below will output the minimum distance between vertex 1 and surface 2.

Measure between vertex 1 surface 2

Measure Small

Measure Small {Length|Area|Volume|All} {Body|Surface} <id_list>

The **Measure Small** command locates all of the lengths, areas, or volumes smaller than the **Measure Small Tolerance** setting. Entities meeting the small tolerance criteria are listed in the output window and typically highlighted in the view port. The following two commands set the small tolerance to 0.1 and output all of the curves within body 1 with lengths at or below the small tolerance.

set measure small tolerance 0.1

measure small length body 1

Geometry

- CUBIT Geometry Formats
- Geometry Creation
- Geometry Transforms
- Geometry Booleans
- Geometry Decomposition
- Geometry Cleanup and Defeaturing
- Geometry Imprinting and Merging
- Virtual Geometry
- Geometry Orientation
- Geometry Groups
- Geometry Attributes
- Geometry Deletion

CUBIT usually relies on the ACIS solid modeling kernel for geometry representation; there is also mesh-based geometry, and a Granite port for Pro Engineer files. Other solid model kernels are planned. Geometry is imported or created within CUBIT. Geometry is created bottom-up or through primitives. CUBIT imports ACIS SAT files. CUBIT can also read STEP, IGES, and FASTQ files and convert them to the ACIS kernel. SolidWorks, AutoCAD, and some other commercial CAD systems can write SAT files directly.

Once in CUBIT, an ACIS model is modified through booleans. Without changing the geometric definition of the model, the topology of the model may be changed using virtual geometry. For example, virtual geometry can be used to composite two surfaces together, erasing the curve dividing them.

Sometimes, an ACIS model is poorly defined. This often happens with translated models. The model can be healed inside CUBIT.

CUBIT Geometry Formats

- ACIS
- Granite
- Mesh-Based Geometry

Setting the Geometry Kernel

The geometry kernel can be switched between ACIS, Mesh-Based Geometry, and Granite from the command line using the following command:

```
set geometry engine {acis|facet|granite}
```

The geometry engine will automatically be set when importing a model.

Terms

Before describing the functionality in CUBIT for viewing and modifying solid geometry, it is useful to give a precise definition of terms used to describe geometry in CUBIT. In this manual, the terms topology and geometry are both used to describe parts of the geometric model. The definitions of these terms are:

Topology: the manner in which geometric entities are connected within a solid model; topological entities in CUBIT include vertices, curves, surfaces, volumes and bodies.

Geometry: the definition of where a topological entity lies in space. For example, a curve may be represented by a straight line, a quadratic curve, or a b-spline. Thus, an element of topology (vertex, curve, etc.) can have one of several different geometric representations.

Topology

Within CUBIT, the topological entities consist of vertices, curves, surfaces, volumes, and bodies. Each topological entity has a corresponding dimension, representing the number of free parameters required to define that piece of topology. Each topological entity is bounded by one or more topological entities of lower dimension. For example, a surface is bounded by one or more curves, each of which is bounded by one or two vertices.

Bodies and Volumes

A CUBIT Body is defined as a collection of other pieces of topology, including curves, surfaces and volumes. The use of Body is not required, and is in fact deprecated in favor of using Volume. Bodies may still be used for grouping volumes, but it is suggested to use Groups instead.

Although a Body may contain groups of Surfaces or Volumes, for most practical purposes within the CUBIT environment, a single Volume or Surface will belong to a single Body. For typical three-dimensional models, this means that there should be one Body for every Volume in the model, where the default Body ID is the same as the Volume ID. For this reason, in many instances the term Volume and Body are used interchangeably, although it is more consistent to always refer to Volumes and Volume IDs, and only use Bodies when absolutely necessary.

Non-Manifold Topology

In many applications, the geometry consists of an assembly of individual parts, which together represent a functioning component. These parts often have mating surfaces, and for typical analyses these surfaces should be joined into a single surface. This results in a mesh on that surface which is shared by the volume meshes on either side of the shared surface. This configuration of geometry is loosely referred to as **non-manifold topology**.

ACIS Geometry Kernel

ACIS is a proprietary format developed by Spatial Technologies. CUBIT incorporates the ACIS third party libraries directly within the program. The ACIS third party libraries are used extensively within CUBIT to import, export and maintain the underlying geometric representations of the solid model for geometry decomposition and meshing. There are many ways to get geometry into the ACIS format. ACIS files can be exported directly from several commercial CAD packages, including SolidWorks, AutoCAD, and HP PE/SolidDesigner. Third party ACIS translators are also available for converting from native formats such as Pro Engineer. CUBIT also uses the ACIS libraries for importing IGES and STEP format files.

Importing and creating geometry using the ACIS geometric modeling kernel currently provides the widest set of capabilities within CUBIT. All geometry creation and modification tools have been designed to work directly on the ACIS representation of the model.

Granite Geometry Kernel

Granite is a proprietary third party geometry kernel that is incorporated directly into CUBIT. Granite is distributed through Parametric Technology Corporation, and is the native format of Pro Engineer. Previously, CUBIT could only read Pro/E files that were translated into ACIS formats, but CUBIT can now import Pro/E files directly. Most of the commands that work for ACIS geometry will also work for Pro/E, with a few exceptions, as noted below.

Limitations

Geometry Creation

1. **Create Body from Surfaces**: Cannot create body from surfaces with command: "create body surface <id_range>"
2. **Sweeping Curves**: When creating bodies or surfaces by sweeping curves, all curves must lie in a plane and sweep direction must be orthogonal to that plane.
3. **Sweeping Surfaces**: When sweeping a surface, surface must be planar and sweep direction must be orthogonal to the surface.
4. **Create Surface from Bounding Curves**: To create a surface from a set of bounding curves ("create surface from curve <id_range>"), all curves must lie in the same plane.

5. **Creating Offset Curves**: Granite does not have the ability to extend offset curves to meet each other when a complete or incomplete loop of curves is offset. So the following rules apply to the create curve offset commands:
    - Multiple linear curves cannot be offset in one command
    - Specified curves must lie in a plane
    - Specified curves must form a single connected chain
6. **Extending Curves**: Granite cannot extend multiple linear curves in one command. Granite has no capability to extend offset curves to meet one another.
7. **Multi-volume Bodies**: Multi-volume bodies cannot be produced in Granite.

Imprinting

1. **Surface**: Surface-surface intersections do not cause any imprinting to occur. A curve must lie ON a surface to be imprinted on it.
2. **Hardlines and Hardpoints**: Hardlines and hardpoints cannot be created from an imprint, as granite does not support hardpoints or hardlines.

Decomposition

1. **Webcutting by Sweeping**: Sweep webcutting is not supported.
2. **Webcutting with Loops**: Webcutting with a loop only succeeds when the loop is planar.
3. **Tweak**: Limited support for tweak command.

Miscellaneous Geometry Options

1. **Split Periodic**: Split periodic command not supported (Granite does not support periodic geometry.)
2. **Healing**: Healing commands not supported.
3. **Vertex Removal**: Vertex removal not supported.
4. **Surface Removal**: Removing surfaces forming a closed loop is not supported.
5. **Regularize**: Regularize command not supported.
6. **Validate**: Validate command not supported.
7. **Tweak** -Tweak cone command not currently supported.
8. **Unite** -Unite not supported with sheet bodies.

Attribute Propagation

1. During a decomposition operation, if a curve is split down the middle, into 2 new identical curves, exactly the same as the original curve, attributes that were on the original curve do not get propagated to new curves.

Export

The Granite format supports export to the following file formats.

- IGES files
- STEP files
- ACIS SAT files
- Granite files. Note: These files can only be read into CUBIT. Pro/E cannot read these files.

Import

The Granite format supports import of the following file formats

- Pro/E part files
- Pro/E assembly files
- IGES files
- STEP files
- Granite files exported from cubit
- Granite Neutral files (not tested yet)

Mesh-Based Geometry

In contrast to the ACIS format, Mesh-Based Geometry (MBG) is not a third party library and has been developed specifically for use with CUBIT. Most of CUBIT's mesh generation tools require an underlying geometric representation. In many cases, only the finite element model is available. If this is the case, CUBIT provides the capability to import the finite element mesh and build a complete boundary representation solid model from the mesh. The solid model can then be used to make further enhancement to the mesh. While the underlying ACIS geometry representation is typically non-uniform rational b-splines (NURBS), Mesh-Based Geometry uses a *facetted* representation. Mesh-Based Geometry can be generated by importing either an Exodus II format file or a facet file.

- Creating Mesh-Based Geometry Models
- Improving Mesh-Based Geometry Models for Meshing
- Meshing Mesh-Based Models
- Exporting Mesh-Based Geometry

Many of the same operations that can be done with traditional CAD geometry can also be done with mesh-based geometry. While all mesh generation operations are available, only some of the geometry operations can be used. For example, the following can be done with geometric entities that are mesh-based:

- Geometry Transformations
- Merging
- Virtual Geometry Operations

Some operations that are not yet available with mesh-based geometry include:

- Booleans
- Geometry Decomposition
- Geometry Clean-Up

Creating Mesh-Based Geometry Models

Mesh based geometry models can be created in one of two ways

- Importing Exodus II files
- Importing facet files

While both of these methods create geometry suitable for meshing, there are some significant differences:

Exodus II files

Exodus II contains a mesh representation that may include 3D elements, 2D elements, 1D elements and even 0D elements. It may also contain deformation information as well as boundary condition information. The import mesh geometry command is designed to decipher this information and create a complete solid model, using the mesh faces as the basis for the surface representations. Exodus II is most often used when a solid model that has previously been meshed requires modification or remeshing. Importing an Exodus II file will generate both geometry and mesh entities, assigning appropriate ownership of the mesh entities to their geometry owners. Deleting the mesh and remeshing, refining or smoothing are common operations performed with an Exodus II model.

Facet files

The facet file formats supported by CUBIT are most often generated from processes such as medical imaging, geotechnical data, graphics facets, or any process that might generate discrete data. Importing a facet file will generate a surface representation only defined by triangles. If the triangles in the facet file form a complete closed volume, then a volume suitable for meshing may be generated. In cases where the volume may not completely close or may not be of sufficient quality, a limited set of tools has been provided. In addition to the standard meshing tools provided in CUBIT, it is also possible to use the triangle facets themselves as the basis for an FEA mesh.

Improving Mesh-Based Geometry Models for Meshing

In many cases, the triangulated representations that are provided from typical imaging processes are not of sufficient quality to use as geometry representations for mesh generation. As a result, CUBIT provides a limited number of tools to assist in cleaning up or repairing triangulated representations.

1. Using tolerance on STL files

Stereolithography (STL) files, in particular, can be problematic. The import mechanism for STL provides a **tolerance** option to merge near-coincident vertices.

2. Using the stitch option on AVS and facet files

The stitch option on the import facets|avs command provides a way to join triangles that otherwise share near-concident vertices and edges. This is useful for combining facet-based surfaces to generate a water-tight model.

3. Using the improve option on facet files.

The improve option on the **import facets** command will collapse short edges on the boundary of the triangulation. This option improves the quality of the boundary triangles.

4. Smoothing faceted surfaces.

Individual triangles in a faceted surface representation may be poorly shaped. Just like mesh elements may be smoothed, facets may also be smoothed in CUBIT using the following command

Smooth <surface_list> facets [iterations <value>] [free] [swap]

To use this command, the surface cannot be meshed. Facet smoothing consists of a simple Laplacian smoothing algorithm which has additional logic to make sure it does not turn any of the triangles in-side out. It also determines a local surface tangent plane and projects the triangle vertices to this plane to ensure the volume will not "shrink". The **iterations** option can be used to specify the number of Laplacian smoothing operations to perform on each facet vertex (The default is 1).

The **free** option can be used to ignore the tangent plane projection. Used too much, the **free** option can collapse the model to a point. One of two iterations of this option may be enough to clean up the triangles enough to be used for a finite element mesh.

The **swap** option can be used to perform local edge swap operations on the triangulation. The quality of each triangle is assessed and edges are swapped if the minimum quality of the triangles will improve.

5. Creating a thin offset volume

Offset surfaces may be generated from an existing facet-based surface. This would be used in cases where a thin membrane-like volume might be required where only a single surface of triangles is provided. This command may be accomplished by using the standard **create body offset** command

The result of this command is a single body with an inside and outside surface separated by a small distance which is generally suitable for tet meshing. This command is currently only useful for small offsets where self-intersections of the resulting surface would be minimal. It is most useful for bodies that may be initially composed of a single water-tight surface.

6. Creating volumes from surfaces

A mesh-based geometry volume can be created from a set of closed surfaces. This can be accomplished in the same manner as the standard create body surface command

Create Body Surface <surface_id_range>

This command is limited to surfaces that match triangles edges and vertices at their boundary. The command will internally merge the triangles to create a water-tight model that would generally be suitable for tet meshing.

Meshing Mesh-Based Models

Mesh-Based models may be meshed just like any other geometry in CUBIT by first setting a scheme, defining a size and using the mesh command. This standard method of mesh generation can be somewhat time consuming and error prone for complex facet models with thousands of triangles. CUBIT also provides the option of using the facets themselves as a surface triangle mesh, or as the input to a tetrahedral mesher. This may be accomplished with one of two options:

Mesh <entity_list> from facets

This command will generate triangular finite elements for each facet on the surface. If the **entity_list** is composed of one or more volumes, then the tetrahedral mesh will automatically fill the interior. This method is useful when further cleanup and smoothing operations are needed on the triangles after import.

Import facets <filename> make_elements

The make_elements on the import facets command will generate the triangular finite elements on the surface at the time the facets are read and created. This option is useful if no further modifications to the facets are necessary.

Creating triangular finite elements in this manner can greatly speed up the mesh generation process, however it is limited to non-manifold topology. If the triangular elements are to be used for tetrahedral meshing (ie. all edges of the triangulation should be connected to no more than two triangles)

Exporting Mesh-Based Geometry

Mesh-Based geometry models and their mesh may be exported by one of the following methods:

- Exporting to an Exodus II File
- Exporting to a facet file

Exodus II

Exporting to an Exodus II file saves the finite element mesh along with any boundary conditions placed on the model. It will not save the individual facets that comprise the mesh-based geometry surface representation. Importing an Exodus II file saved in this manner will regenerate the surfaces only to the resolution of the saved mesh.

Facet files

CUBIT also provides the option to save just the surface representation to a facet or STL file. The following commands can be used for saving facet or STL files:

Export Facets 'filename' <entity_list> [overwrite]

Export STL [ASCII|binary] 'filename' <entity_list> [overwrite]

These commands provide the option of saving specific surfaces or volumes to the facet file. If no entities are provided in the command, then all surfaces in the model will be exported to the file. The **overwrite** option forces a file to overwrite any file of the same name in the current working directory.

Geometry Creation

There are three primary ways of creating geometry for meshing in CUBIT. First, CUBIT provides many geometry primitives for creating common shapes (spheres, bricks, etc.) which can then be modified and combined to build complex models. Secondly, geometry can be imported into CUBIT. Finally, geometry can be defined by building it from the "bottom up", creating vertices, then curves from those vertices, etc. Two of these three methods for creating geometry in CUBIT will be described in detail in this section.

- Bottom-Up Geometry Creation
- Geometric Primitives

Bottom-Up Geometry Creation

CUBIT supports the ability to create geometry from a collection of lower order entities. This is accomplished by first creating vertices, connecting vertices with curves and connecting curves into surfaces. Currently bodies or volumes may not be constructed by stitching a set of surfaces together, however surfaces may be swept or rotated to create bodies or volumes. Existing geometry may be combined with new geometry to create higher order entities. For example, a new surface can be created using a combination of new curves and curves already extant in the model. Commands and details for creating each type of geometry entity are given below.

The following describes each of the basic entities that can be generated with CUBIT using the bottom-up approach

- Creating Vertices
- Creating Curves
- Creating Surfaces
- Creating Bodies

Creating Vertices

The basic commands available for creating new vertices directly in CUBIT are:

- XYZ location
- On Curve - Fraction
- On Curve - General
- From Vertex
- At Arc
- At Intersection

1. XYZ location: The simplest form of this command is to specify the XYZ location of the vertex. It can also be created lying on a curve or surface in the geometric model by specifying the curve or surface id; the position of the vertex will be the point on the specified entity which is closest to the position specified on the command. With all of these commands, the user is able to specify the color of the vertex.

Create Vertex <x><y><z> [on [Curve | Surface] <id>] [Color <color_name>]

2. On Curve - Fraction: A vertex can be positioned a certain fraction of the arc length along a curve using the second form of the command.

Create Vertex <fraction> from Vertex <id> on Curve <id>[Color <color_name>]

3. On Curve - General: A more general purpose form of the command is also available for creating vertices on curves:

Create Vertex On Curve <id_list> { MIDPOINT | Start | End | Fraction <val 0.0 to 1.0> [From Vertex <id> | Start|End] | Distance <val> [From {Vertex|Curve|Surface} <id> | Start|End] | {{Close_To|At} Location {options} | Position <xval><yval><zval>|{Node|Vertex} <id>} | Extrema [Direction] {options} [Direction {options}] [Direction {options}] | Segment <num_segs> | Crossing {Curve|Surface} <id_list> [Bounded|Near] } [Color <color_name>]

It allows the vertex to be created at a fractional distance along the curve, at an actual distance from one of the curves ends, at the closest location to an xyz position or another vertex, or at a specified distance from a vertex, curve or surface. You can also preview the location first with the command Draw Location On Curve... (where the rest of the command is identical to the Create Vertex form).

4. From Vertex: Create a vertex from an existing vertex.

Create Vertex from Vertex <id_list> [on {curve|surfce} <id>] [Color <color_name>]

If 'on curve|surface' option is used, the vertex is positioned on that curve or surface. When the 'on curve|surface' is not used, the new vertex is positioned on the existing vertex.

5. At Arc: Another form simply creates vertices at arc or circle centers.

Create Vertex Center Curve <id_list> [Color <color_name>]

6: At Intersection: The last form creates vertices at the intersection of two curves. If the *bounded* qualifier is used, the vertices are limited to lie on the curves, otherwise the extensions of the curves are also used to calculate the intersections. The *near* option is only valid for straight lines, where the closest point on each curve is created if they do not actually intersect (resulting in two new vertices).

Create Vertex AtIntersection Curve <id1> <id2> [bounded] [near] [Color <color_name>]

Creating Curves

Curves are created by specifying the bounding lower-order topology (i.e. the vertices) and the geometry (shape) of the curve (along with any parameters necessary for that geometry). There are several forms of this command:

- Straight
- Parabolic, Circular, Ellipse
- Spline
- Copy
- Arc Three
- Arc Center Vertex
- From Vertex Onto Curve
- Offset
- From Mesh Edges
- Close_To
- Surface Intersection
- Projecting onto Surface

1. Straight: The first form of the command creates a straight line or a line lying on the specified surface. If a surface is used, the curve will lie on that surface but will not be associated with the surface's topology.

Create Curve [Vertex] <vertex_id> [Vertex] <vertex_id> [On Surface <surface_id>]

Straight curves can be created using an axis. The syntax is as follows:

Create Curve Axis {options}

The length of the axis must be specified. Go to Location, Direction, and Axis Specification to see the axis command description.

Additionally, several connected straight curves can be created with a single command. The syntax for the polyline command is as follows:

Create Curve Polyline Location {options} Location {options} ...

Notice that two or more locations are used to define a polyline. See Location, Direction, and Axis Specification for the location command description.

2. Parabolic, Circular, Ellipse: The Parabolic option creates a parabolic arc which goes through the three vertices. The Circular and Ellipse options create circular and elliptical curves respectively that go through the first and last vertices.

Create Curve [Vertex] <vertex_id> [Vertex] <vertex_id> [Vertex] <vertex_id>
[Parabolic|Circular|Ellipse]

3. Spline: The spline form of the command creates a spline curve that goes through the all input vertices or locations. To create a curve from a list of vertices use the syntax shown below. The **delete** option will remove all of the intermediate vertices used to create the spline leaving only the end vertices.

Create Curve [Vertex] <vertex_id_list> [Spline] [delete]

Additionally, spline curves can be created by inputting a list of locations. Where the spline will pass through all of the specified locations. The syntax is shown below:

Create Curve Spline {List of locations}

See Location, Direction, and Axis Specification to view the location specification syntax.

4. Copy: This command actually copies the geometric definition in the specified curve to the newly created curve. The new curve is free floating.

Create Curve from Curve <curve_id>

5. Arc Three: The following command creates an arc either through 3 vertices or tangent to 3 curves. The *Full* qualifier will cause a complete circle to be created.

Create Curve Arc Three {Vertex|Curve} <id_list> [Full]

6. Arc Center Vertex: The next form of the command creates an arc using the center of the arc and 2 points on the arc. The arc will always have a radius at a distance from the center to the first point, unless the *Radius* value is given. Again, the *Full* qualifier will cause a complete circle to be created.

Create Curve Arc Center Vertex <center_id> <end1_id> <end2_id> [Radius <value>] [Full] [Normal <x> <y> <z> ***Needed when points colinear]

Note: Requires 3 Vertices - first is center, other two are on the arc

7. From Vertex Onto Curve: The following command will create a curve from a vertex onto a specified position along a curve. If none of the optional parameters are given, the location on the curve is calculated as using the shortest distance from the start vertex to the curve (i.e., the new curve will be normal to the existing curve).

Create Curve From Vertex <vertex_id> Onto Curve <curve_id> [Fraction <f> | Distance <d> | Position <xval><yval><zval> | Close_To Vertex <vertex_id> [[From] Vertex <vertex_id> (optional for 'Fraction' & 'Distance')]] [On Surface <surface_id>]

Note: Default = Normal to the Curve

8. Offset: The next command creates curves offset at a specified distance from a planar chain of curves. The direction vector is only needed if a single straight curve is given. The offset curves are trimmed or extended so that no overlaps or gaps exist between them. If the curves need to be extended the extension type can be *Rounded* like arcs, *Extended* tangentially (the default -straight lines are extended as straight lines and arcs are extended as arcs), or extended *naturally*.

Create Curve Offset Curve <id_list> Distance <val> [Direction <x> <y> <z>] [Rounded|EXTENDED|Natural]

Note: Direction is optional for offsets of individual straight curves only

In all cases, the specified vertices are not used directly but rather their positions are used to create new vertices.

9. From Mesh Edges: This commands creates a curve from an existing mesh given a starting node and an adjacent edge.

Create Curve From Mesh Node <id> Edge <id> [Length <val>]

The adjacent edge indicates which direction to propagate the curve.
The curve will be composed of mesh edges up to the specified length.
If no length is specified the curve will propagate as far as the boundary of the mesh. Figure 1 shows a example of a curve generated from the mesh.

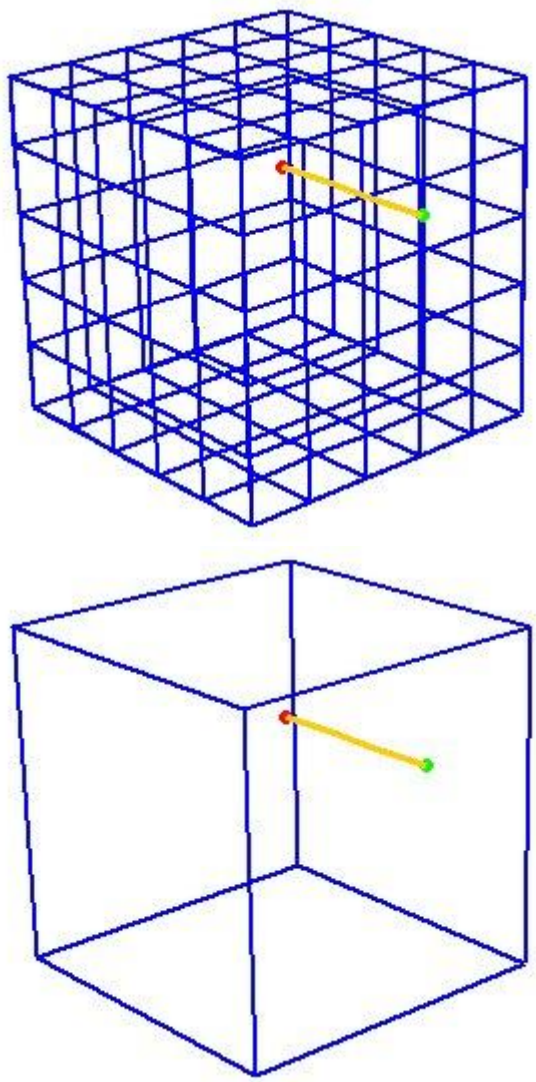


Figure 1. Example of curve created from mesh

The underlying geometry kernel used for this command is Mesh-Based geometry. The new curve will also be meshed with the edges it was propagated through. A related command for assigning mesh edges directly to a mesh block is the Rebar command. See Element Block Specification for more details.

Note: Full hexes or full tets must be used to propagate the curves through the interior of volume.

10. Close_To This option takes two geometric entities and creates the shortest possible curve between the two entities at the location where the two entities are the closest. The two entities may NOT intersect. If two vertices are given, the command will create a straight line between the two vertices.

Create Curve Close_To {Vertex|Curve|Surface|Volume|Body} <id_1>
{Vertex|Curve|Surface|Volume|Body} <id_2>

11. Surface Intersection The following command creates curves at surface intersections. Multiple curves can be created from a single command.

Create Curve Intersecting Surface <id_list>

12. Projecting onto a Surface The project command allows you to make an imprint of a surface or set of curves onto another surface. The command syntax is as follows:

Project {Curve|Surface} <id_list> onto Surface <surface_id> [Imprint [keepcurve] [keepbody]]

The command takes a list of curves or surfaces, and a projection surface. If a list of curves is given, the result will be the creation of a set of free curves on top of the projection surface. If a list of surfaces is given, the result will be the same as selecting the curves that bound the surface (i.e. a group of free curves on the projecting surface).

The imprint option will imprint the resulting projected curves onto the projection surface. If this option is NOT given, the new curves will lie coincident to the surface, but will not be part of the surface. Imprinting changes the topology of the projection surface. Keepcurve option retains the new curves as both free curves, and curves in the projection surface. The keepbody option retains the original body under the new imprinted body.

Creating Surfaces

There are two major ways to create surfaces in CUBIT. First, surfaces can be created in CUBIT by fitting an analytic or spline surface over a set of bounding curves. In this case, the curves must form a closed loop, and only one loop of curves may be supplied. The second method, is by sweeping a curve about an axis, along a vector, or along another curve. The result of these surface creation commands is a "sheet body" or a body that has zero measurable volume (it does however have a volume entity). Booleans and special webcutting commands may be used with to decompose this body or to use it for decomposing other bodies. Booleans can be used to cut holes out of these surfaces.

The following options may be used for creating a surface in CUBIT.

- Bounding Curves
- Bounding Vertices or Nodes
- Copy
- Extended Surface
- Planar Surface
- Net Surface
- Offset
- Skinning
- Sweeping of Curves
- Midplane
- Weld Profile
- Meshed Entities

1. Bounding Curves: The first form of this command produces an analytic or spline surface fit to cover the bounding curves.

Create Surface Curve <curve_id_1> <curve_id_2> <curve_id_3>...

Another version of this command creates a surface from a set of bounding curves that all lie on one surface. If the curves are selected they must lie on the surface, and they must create a closed loop. The **On Surface** option forces the surface to match the geometry of the underlying surface exactly.

Create Surface Curve <id_list> On Surface <surface_id>

2. Bounding Vertices or Nodes: The second form of this command uses vertices to fit an analytic spline surface. The **On Surface** option creates the surface from a set of nodes and vertices that all lie on one surface and restrains the surface to match the geometry of the underlying surface. The project option will project the nodes or vertices to the specified surface.

Create Surface [Node|Vertex| <id_list> [On Surface <surface_id> {Project}]

3. Copy: The next form creates a surface using the same geometric description of the specified surface. The new surface will be a stand-alone sheet body that is geometrically identical to the user supplied surface.

Create Surface from Surface <surface_id>

4. Extended Surface: The fourth form of the command creates a surface that is extended from a given surface. The specified surface's geometry is examined and extended out "infinitely" relative to the current model in CUBIT (i.e. extended to just beyond the bounding box of the entire model). The given surfaces are extended as shown in the following table.

Create Surface extended from Surface <surface_id>

Table 1. Surface Extension Results

Surface Type	Resulting Extended Surface
Spherical	Shell of Full Sphere
Planar	Plane of infinite size relative to model
Toroidal	Shell of Full Torus
Conical, cone, cylinder...	Shell of outside conic axially aligned with given conic of infinite height relative to model
Spline	Surface is extended to extents of the spline definition. This may not be any further than the surface itself, so caution should be used here.

5. Planar Surface: The following commands create ***planar*** surfaces. The first passes a plane through 3 vertices, the second uses an existing plane, the third creates a plane normal to one of the global axes, and the fourth creates a plane normal to the tangent of a curve at a location along the curve. By default, the commands create the surface just large enough to intersect the bounding box of the entire model with minimum surface area. Optionally, you can give a list of bodies to intersect for this calculation. You can also extend the size of the surface by either a percentage distance or an absolute distance of the minimum area size. The plane can be previewed with the command Draw Plane [with]... (where the rest of the command is the same as that to create the surface).

Create Planar Surface [with] Plane Vertex <v1_id> [vertex] <v2_id> [vertex] <v3_id> [intersecting] Body <id_range>] [extended percentage|absolute <val>]

Create Planar Surface [with] Plane Surface <surface_id> [intersecting] Body <id_range>] [extended percentage|absolute <val>]

Create Planar Surface [with] Plane {xplane|yplane|zplane} [offset <val>] [intersecting] Body <id_range>] [extended percentage|absolute <val>]

Create Planar Surface [with] Plane Normal To Curve <curve_id>{fraction <f> | distance <d> | position <xval><yval><zval> | close_to vertex <vertex_id>} [[from] Vertex <vertex_id> (optional for 'fraction' & 'distance')] [intersecting] Body <id_range>] [extended percentage|absolute <val>]

6. Net Surface: *Net surfaces* can be created with two different commands. A net surface passes through a set of curves in the u-direction and a set of curves in the v-direction (these u and v curves would looked like a mapped mesh). The first form of the command uses curves to create the net surface. The curves must pass within tolerance of each other to work. The second form uses a mapped mesh to create the surface. The mapped mesh can be of a single surface or a collection of mapped or submapped surfaces that form a logical rectangle. By default net surfaces are healed to take advantage of any possible internal simplification.

Create Surface Net U Curve <id_list> V Curve <id_list> [Tolerance <value>] [HEAL|noheal]

Create Surface Net [From] [Mapped] Surface <id_list> [Tolerance <value>] [HEAL|noheal]

A suggested geometry cleanup method is to use a virtual composite surface to map mesh a set of complicated surfaces then create a net surface from this mesh. Then the original surfaces can be removed with the *noextend* option and the new net surface combined back onto the body.

7. Offset: The following command creates surfaces ***offset*** from existing surfaces at the specified distance. The surfaces are not guaranteed to be extended or trimmed to share boundaries; however they are generally close.

Create Surface Offset [From] Surface <id_list> Distance <val>

8. Skinning: The following command creates a ***skin*** surface from a list of curves. An example of a skin surface is to create a surface through a set of parallel lines.

Create Surface Skin Curve <id_list>

9. Sweeping of Curves: A curve or a set of curves can be swept along a path to create new surfaces. The path may be specified as an axis and angle, a vector and distance, or by indicating another curve or set of contiguous curves. The following commands show the options available:

Sweep Curve <curve_id_range> { axis <xpoint ypoint zpoint xvector yvector zvector> |xaxis | yaxis | zaxis } angle <degrees> [steps <Number_of_sweep_steps>] [draft_angle <degrees>] [draft_type <integer>] [make_solid] [rigid]

Sweep Curve <curve_id_range> vector <xvector yvector zvector> [distance <distance>] [draft_angle <degrees>] [draft_type <integer>] [rigid]

Sweep Curve <curve_id_range> along curve <refcurve_id_range> [draft_angle <degrees>] [draft_type <integer>] [rigid]

In the first command, the steps options provides a way of faceting the sweep, so instead of a smooth round sweep, there are facets to the surface. The make_solid option closes the newly-created surface to the axis, so that a solid is created instead of a surface.

The other options are as follows:

draft_angle: determines how much drafting in of the surface is desired
draft_type:

0 => extended (draws two straight tangent lines from the ends of each segment until they intersect)

1 => rounded (create rounded corner between segments)

2 => natural (extends the shapes along their natural curve) ***

rigid: normally the curve will rotate to maintain its original orientation to the sweep path. The rigid option disallows this rotation.

10. Midplane: Surfaces may be created midway between two planar surfaces using the following command:

Create Midplane Body <id> Surface <id1> <id2>

Specify the IDs of two planar surfaces. (This command does not support non-planar surfaces). The resulting surface will be trimmed by the specified body. **Figure 1** shows an example of two surfaces created using the **Create Midplane** command.

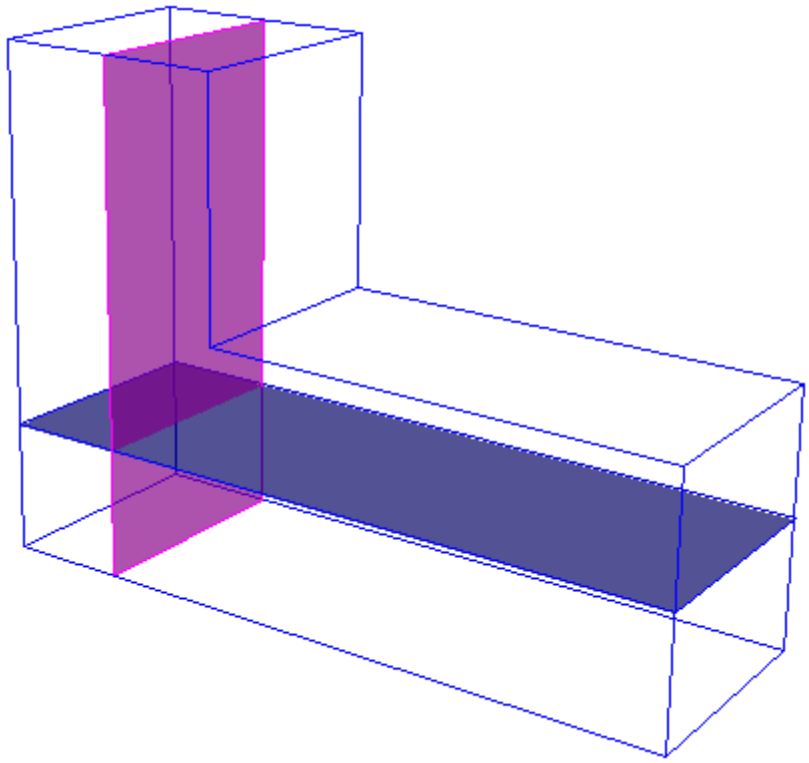


Figure 1. Surface created with the Create Midplane command

11. Weld Profile: Surfaces may be created by specifying a weld profile using the following command:

Create Surface Weld [Root] Location {options} Weld Surface <id_list> Length <val> [<val2>]

Weld surfaces can be used to create a simulated welded joint by sweeping the surface along the root curve and uniting the new body to the model. An example of the command is illustrated below. For a detailed description of the location specifier see Location Direction, and Axis Specification.

create surface weld root location vertex 25 weld surface 13 14 length 2

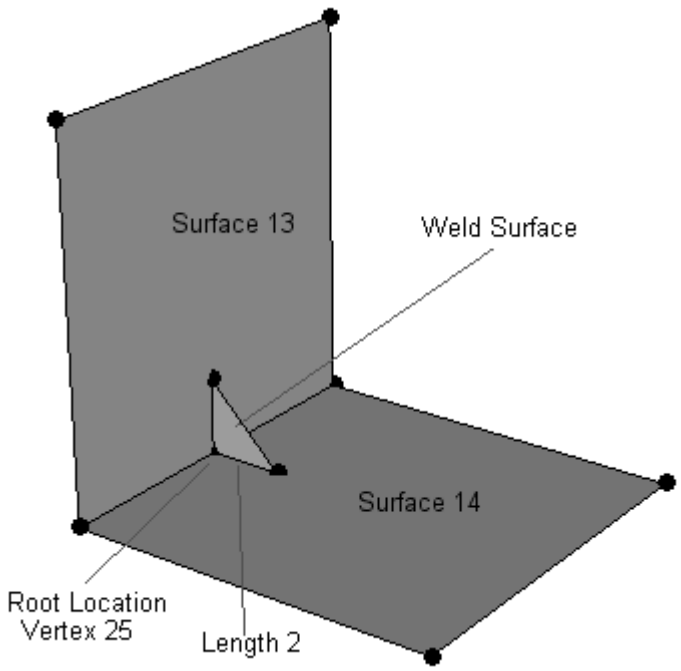


Figure 2. Weld Profile surface with length and root specifications

12. Creating A Surface From Mesh Entities: Surfaces may be created from the boundaries of meshed volumes, surfaces, and/or from individual quadrilateral mesh elements. The individual option makes it so you can enter multiple surfaces at once, and not have them merged together into a larger surface, but instead retain their own original boundaries.

Create Acis [From] {Surface <id_range> | Volume <id_range> | Face < id_range> [Individual]}

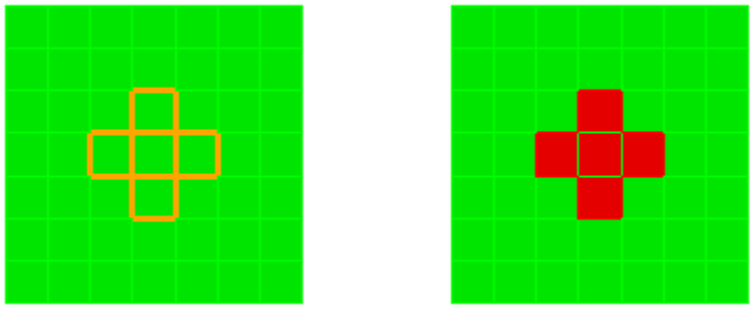


Figure 3. Acis Surface created from a Set of Quadrilaterals

Creating Bodies

Currently, CUBIT can create volumes:

1. from surfaces by sweeping a single surface into a 3D solid,
2. by offsetting an existing volume,
3. by sweeping a curve around an axis.
4. by stitching together surface that can form a closed volume.
5. by lofting from one surface to another surface.

6. by thickening a surface body.

Sweeping of planar surfaces, belonging either to two- or three-dimensional bodies, is allowed, and some non-planar faces can be swept successfully, although not all are supported at this time. The following methods for generating volumes are described:

- Sweep Surface Along Vector
- Sweep Surface About Axis
- Sweep Surface Along Curve
- Offset
- Sweep Curve About Axis
- Stitch Surfaces Together
- Loft Surfaces Together
- Thicken Surfaces

There are four forms of the sweep command; the syntax and details for each are given below. In each form, the optional **draft_angle** parameter specifies the angle at which the lateral faces of the swept solid will be inclined to the sweep direction. It can also be described as the angle at which the profile expands or contracts as it is swept. The default value is 0.0. The optional **draft_type** parameter is an ACIS-related parameter and specifies what should be done to the corners of the swept solid when a non-zero draft angle is specified. A value of 0 is the default value and implies an extended treatment of the corners. A value of 1 is also valid and implies a rounded (blended) treatment of the corners.

1. Sweep Surface Along Vector: Sweeps a surface a specified distance along a specified vector. Specifying the distance of the sweep is optional; if this parameter is not provided, the face is swept a distance equal to the length of the specified vector.

Sweep surface {<surface_id_range> | all} vector <x_vector y_vector z_vector> [distance <distance_value>] [draft_angle <degrees>] [draft_type <0 | 1>]

2. Sweep Surface About Axis: Sweeps a surface about a specified vector or axis through a specified angle. The axis of revolution is specified using either a starting point and a vector, or by a coordinate axis. This axis must lie in the plane of the surfaces being swept. The steps parameter defaults to a value of 0 which creates a circular sweep path. If a positive, non-zero value (say, n) is specified, then the sweep path consists of a series of n linear segments, each subtending an angle of [(sweep_angle) / (steps-1)] at the axis of revolution.

Sweep surface {<surface_id_range> | all} axis {<xpoint ypoint zpoint xvector yvector zvector> | xaxis | yaxis | zaxis} angle <degrees> [steps <number_of_sweep_steps>] [draft_angle <degrees>] [draft_type <0 | 1>]

Specifying multiple surfaces that belong to the same body will not work as expected, as ACIS performs the sweep operation in place. Hence, if a range of surfaces is provided, they ought to each belong to different bodies.

3. Sweep Surface Along Curve: This command allows the user to sweep a planar surface along a curve:

Sweep Surface <surface_id_range> Along Curve <curve_id> [draft_angle <degrees>] [draft_type <0 | 1 | 2>]

One of the ends of the curve must fall in the plane of the surface and the curve cannot be tangential to the surface. Sweep along curve also supports an additional draft type "2" which implies a "natural" extension of the corners from their curves.

The sweep operations have been designed to produce valid solids of positive volume, even though the underlying solid modeling kernel library that actually executes the operation, ACIS, allows the generation of solids of negative volume (i.e., voids) using a sweep.

4. Offset: The following command creates a body offset from another body at the specified distance. The new surfaces are extended or trimmed appropriately. A positive distance results in a larger body; a negative distance in a smaller body.

Create Body Offset [from] Body <id_range> Distance <value>

This option is also available for limited cases for facet-based surfaces.

5.Sweep Curve About Axis: Sweeps a curve or set of curves about a given axis through a specified angle. The axis is specified the same as in the Sweep Surface About Axis command. The steps, draft_angle, and draft_type options are the same as are described above. To create the solid, the make_solid option must be specified, otherwise a surface will be created, rather than a solid. If the rigid option is specified, then the curve or set of curves will remain oriented as originally oriented, rather than rotating about the axis.

> Sweep Curve <curve_id_range> { axis <xpoint ypoint zpoint xvector yvector zvector> | xaxis | yaxis | zaxis } angle <degrees> [steps <Number_of_sweep_steps>] [draft_angle <degrees>] [draft_type <integer>] [make_solid] [rigid]

6. Stitch Surfaces Together: A body can be created from various surfaces that form a closed volume with command below. The geometry must be ACIS-type geometry (ie. can be imported from IGES, STEP or fastq files)

> Create Body Surface <surface_id_range>

This option is also available for limited cases for facet-based surfaces.

7. Loft Surfaces Together: A body can be "lofted" between two surfaces to form a new body. Surfaces from solid bodies and sheet bodies may be used to create a loft body. In order to create the loft body, two surfaces coincident to the input surfaces are created. The loft body is extruded along the shortest path between the corresponding vertices that define the shapes of the two copied surfaces. This new body is solid. The surfaces used to create the loft body are unchanged.

> Create {Body|Volume} Loft Surface <surf1> <surf2> [Takeoff1 <value>] [Takeoff2 <value>] [arc_length {true|FALSE}] [twist {TRUE|false}] [align_direction {TRUE|false}] [perpendicular {TRUE|false}] [simplify {true|FALSE}]

It is recommended that lofting only be attempted between similar surfaces. For example, lofting from a trapezoidal surface (whose shape is defined by four end vertices) to a triangular surface (whose shape is defined by three end vertices) will force the lofting function to transform the cross-section of the loft body in mid-extrusion, often with poor results (e.g., a skewed or self-intersecting loft body). Attempting to loft between nearly perpendicular surfaces generally produces poor results as well.

Lofting can be used to split a body in order to create a more structured mesh. Figure 1 below shows a single volume swept from a large paved surface. Figure 2 shows this same volume after surfaces defined on the source and target surfaces have been used to create a loft body. This original body was chopped with the loft body. The resulting two bodies were merged. The yellow volume was swept as the volume in Figure 1 was but the purple volume was submapped, producing a much more structured mesh overall.

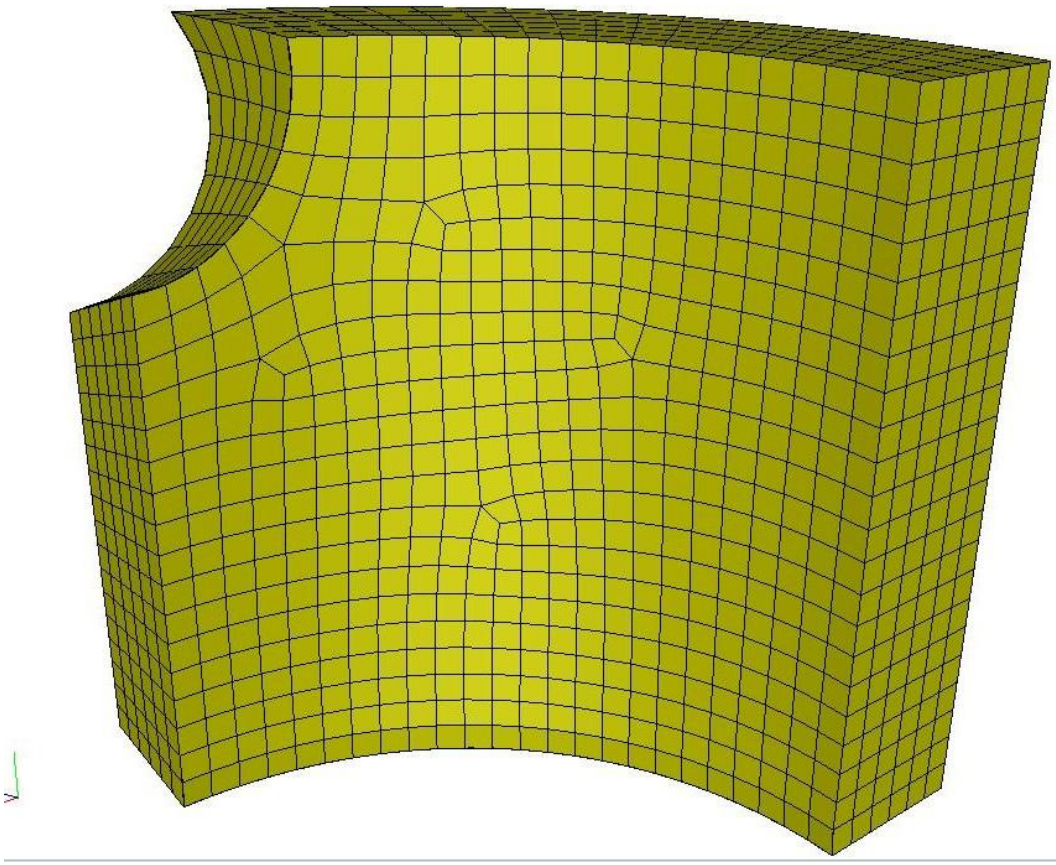


Figure 1. Mesh before loft. Single swept volume with a large paved face.

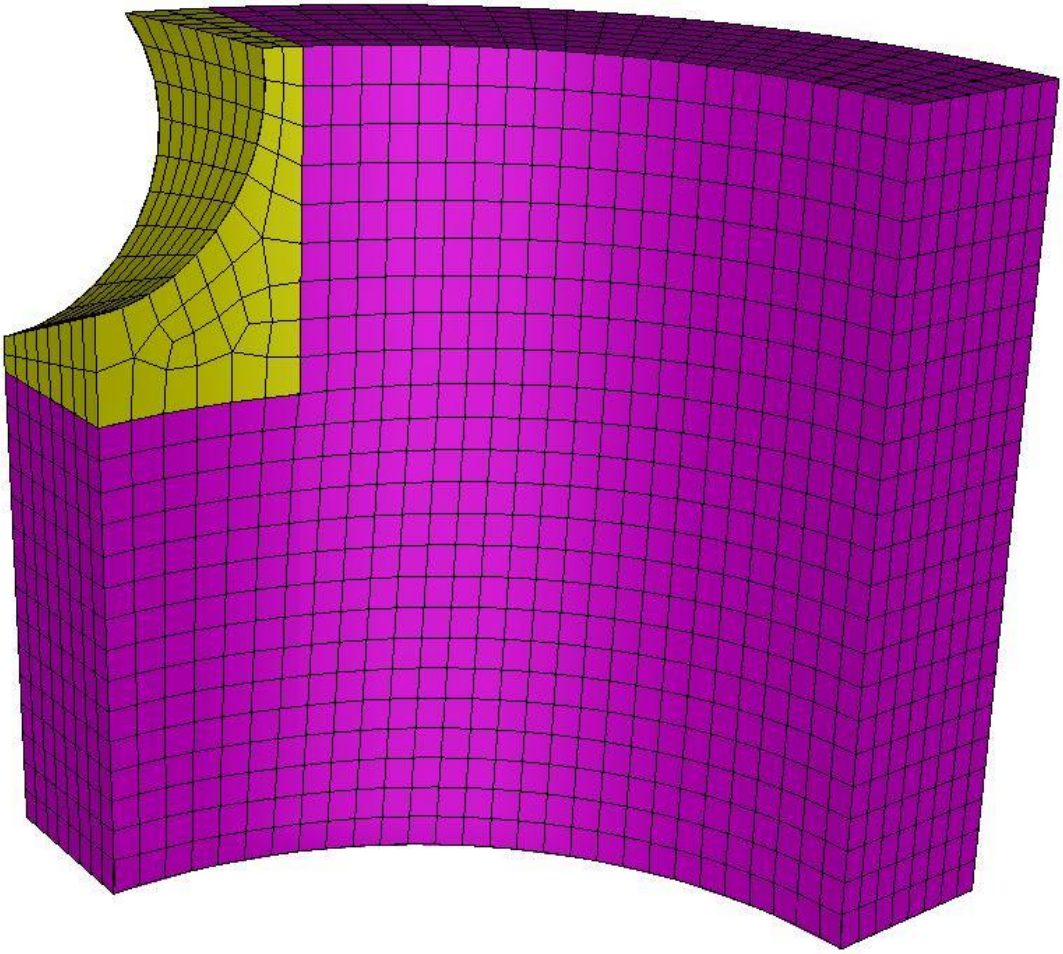


Figure 2. Mesh after loft. The yellow volume is paved and the purple volume is submapped.

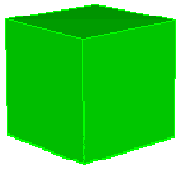
8. Thicken Surfaces: A surface body can be thickened to create a volume body. The surface can be thickened in both directions using the "both" keyword, thickened in the direction of surface normal using a positive depth, or thickened in the opposite direction using a negative depth. To thicken multiple surfaces, all surface normals must be consistent.

Thicken [body] <body_list> depth <depth> [both]

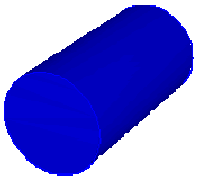
Geometric Primitives

The geometric primitives supported within CUBIT are pre-defined templates of three-dimensional geometric shapes. Users can create specific instances of these shapes by providing values to the parameters associated with the chosen primitive. Primitives available in CUBIT include the brick, cylinder, torus, prism, frustum, pyramid, and sphere. Each primitive, along with the command used to generate it and the parameters associated with it, are described next. For some primitives, several options can be used to generate them, and are described as well.

The following Primitives can be generated with CUBIT:

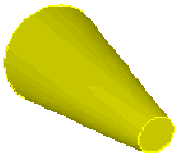


Brick

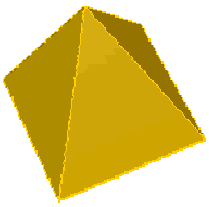


Cylinder

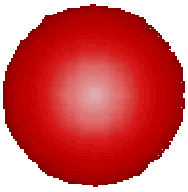
Prism



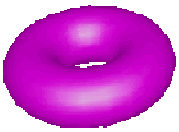
Frustrum



Pyramid



Sphere



Torus

General Notes

- Primitives are created and given an ID equal to one plus the current highest body ID in the model.
- Primitive solids are created with their centroid at the origin or the world coordinate system.
- For primitives with a Height or Z parameter, the axis going through these primitives will be aligned with the Z axis.
- For primitives with a Major Radius and a Minor Radius, the Major Radius will be along the X axis, the Minor Radius along the Y axis.
- For primitives with a Top Radius, this radius will be that along the X axis; the Y axis radius will be computed using the Major, Minor and Top Radii given.

Creating Bricks

The brick is a rectangular parallelepiped.

Command

[Create] Brick {Width|X} <width> [{Depth|Y} <depth>] [{Height|Z} <height>] [Bounding Box entity_type <id_range>] [Tight] [[Extended] {Percentage| Absolute} <val>]]

Notes

- A cubical brick is created by specifying only the width or x dimension.
- A brick can be specified to occupy the bounding box of one or more entities, specified on the command line.
- If the **Tight** option is specified with **Bounding Box**, the result is the smallest brick that can contain the entities specified, which is the default behavior of the Bounding Box option.
- If the **Extended** option is specified with **Bounding Box**, the result is a brick that is extended from a "tight" brick by the input percentage or absolute value.
- If a bounding box specification is used in conjunction with any of the other parameters (X, Y or Z), the parameters specified override the bounding box results for that or those dimensions.

Creating Cylinders

The cylinder is a constant radius tube with right circular ends.

Command

[Create] Cylinder [height | z] <val> Radius <val>

[Create] Cylinder [height | z] <val> Major Radius <val> Minor Radius <val>

Notes

- A cylinder may also be created using the frustum command with all radii set to the same value.
- Specifying major and minor radii can produce a cylinder with an oval cross section.

Creating Prisms

The prism is an n-sided, constant radius tube with n-sided planar faces on the ends of the tube.

Command

[Create] Prism [height | z] <z-val> sides <nsides> radius <radius>

Notes

- The radius defines the circumradius of the n-sided polygon on the end caps.
- If a major and minor radius are used, the end caps are bounded by a circum-ellipse instead of a circumcircle.
- The number of sides of a prism must be greater than or equal to three. A prism may also be created using the pyramid command with all radii set to the same value.
- If the **Extended** option is specified with **Bounding Box**, the result is a brick that is extended from a "tight" brick by the input percentage or absolute value.
- If a bounding box specification is used in conjunction with any of the other parameters (X, Y or Z), the parameters specified override the bounding box results for that or those dimensions.

Creating Frustrums

A frustum is a general elliptical right frustum, which can also be thought of as a portion of a right elliptical cone.

Command

[Create] Frustum [height | z] <z-height> Radius <x-radius> [Top <top_radius>]

[Create] Frustum [height | z] <z-height> Major Radius <radius> Minor Radius <radius> [Top <top_radius>]

Notes

- If used, Major Radius defines the x-radius and Minor Radius the y-radius.
- If used, Top Radius defines the x-radius at the top of the frustum; the top y radius is calculated based on the ratio of the major and minor radii.

Creating Pyramids

A pyramid is a general n-sided prism.

Command

[create] pyramid [height | z] <z-height> sides <nsides> Radius <radius>

[create] pyramid [height | z] <z-height> sides <nsides> [major [radius] <x-radius> minor [radius] <y-radius>] [top <top-x-radius>]

Creating Spheres

The sphere command generates a simple sphere, or, optionally, a portion of a sphere or an annular sphere.

Command

[Create] sphere radius <radius> [xpositive] [ypositive] [zpositive] [delete] [inner [radius] <radius>]

Notes

- If Xpositive, Ypositive, and/or Zpositive are used, a sphere which occupies that side of the coordinate plane only is generated, or, if the delete keyword is used, the sphere will occupy the other side of the coordinate plane(s) specified.These options are used to generate hemisphere, quarter sphere or a sphere octant (eighth sphere).
- If the inner radius is specified, a hollow sphere will be created with a void whose radius is the specified inner radius.

Creating Toruses

The torus command generates a simple torus

Command

[create] torus major [radius] <major-radius> minor [radius] <minor-radius>

Notes

- **Minor Radius** is the radius of the cross-section of the torus; **Major Radius** is the radius of the spine of the torus.
- The **minor radius** must be less than the **major radius**.

Geometry Transforms

- Align
- Copy
- Move
- Scale
- Rotate
- Reflect

Bodies can be modified in CUBIT using transform operations, which include align, copy, move, reflect, restore, rotate, and scale. With the exception of the copy operation, transform operations in CUBIT do not create new topology, rather they modify the geometry of the specified bodies. ACIS, Mesh Based Geometry and Virtual Geometry representations may be transformed. If the geometric entity has been meshed, the nodes of the mesh will be transformed along with the geometry. To transform the nodes of a mesh as they are written to the Exodus II mesh file without modifying their location within CUBIT, see Transforming Mesh Coordinates.

Align Command

The align command is a combination of the rotate and move commands. The align command will align the surface of a given volume with any other surface in the model, such that the surface centroids are coincident and the normals are pointing either in the same or opposite direction (depending on their initial alignment). The align command can also align a face of a volume with the xy, yz, and xz planes and the vertices of a volume with the x, y, and z axes.

The syntax of the command to align commands are:

Align Volume <id> Surface <surface_id> with Surface <surface_id>

Align Volume <id> {Surface <surface_id>| Vertex <vertex_id>} {{x|y|z axis}|{xy|xz|yz plane}}

This transformation is useful for aligning surfaces in preparation for geometry decomposition and aligning models for axis-symmetric analysis.

Copy Command

The copy command copies an existing entity to a new entity without modifying the existing entity. A copy can be made of several entities at once, and the resulting new entities can be translated or rotated at the same time. The commands for copying entities are:

{Body|Volume|Surface|Curve} <range> Copy [move {x|y|z} <distance>] [nomesh]

{Body|Volume|Surface|Curve} <range> Copy [move <direction> [distance]] [nomesh]

{Body|Volume|Surface|Curve} <range> Copy [Reflect {x|y|z}] [nomesh]

{Body|Volume|Surface|Curve} <range> Copy [Reflect <x> <y> <z>] [nomesh]

{Body|Volume|Surface|Curve} <range> Copy [Rotate <angle> About {x|y|z}] [nomesh]

{Body|Volume|Surface|Curve} <id_range> Copy [Rotate <angle> About <x> <y> <z>] [nomesh]

{Body|Volume|Surface|Curve} <range> Copy
[Scale <scale> | x <val> y <val> z <val>] [nomesh]

If the copy command is used to generate new entities, a copy of the original mesh generated in the original entity will also be copied directly onto the new entity unless the **nomesh** option is used.

This is currently limited to copies that do not interact with adjacent geometry through non-manifold topology. For details on mesh copies, see the Mesh Duplication documentation.

Move Command

The move command moves a body by a specified offset. The commands to move bodies are:

Body <id_range> [Copy] Move <dx> <dy> <dz>

Body <id_range> [Copy] Move {x|y|z} <distance>

Move Body <id_range> Normal To Surface <id> Distance <val>

Move Body <id_range> XYZ <x_val> <y_val> <z_val>

where <dx> <dy> <dz> and <x_val> <y_val> <z_val> represent relative offsets in the major axis directions. If the copy option is specified, a copy is made and the copy is moved by the specified offset. The Move Normal to Surface option will move a body along an axis defined by the outward-facing surface normal of a specified surface.

Moving Other Geometric Entities

It is also possible to move bodies by specifying one of its child entities. For example, a body can by moved by specifying one of its curves. However, if a lower-order entity is moved, the parent body and all related entities will also be moved. The commands for moving bodies using a child entity are given below. Alternatively, the tweak command can be used to move curves and surfaces without moving the parent body.

Move {vertex|curve|surface|volume|body} <id_range> [Midpoint] Location <x> [<y> [<z>]]

Move {vertex|curve|surface|volume|body} <id_range> location [Midpoint] [x <val>] [y <val>] [z <val>] [except [x] [y] [z]]

Move {vertex|curve|surface|volume|body} <id_range> Normal to Surface <id> Distance <val>

The first form of the command will move the entity to an absolute location. The second form will move the entity by a relative distance in any of the xyz axis directions. "Except" is used to preserve the x, y, or z plane in which the center of the entity lies. The third form of the command will move the body along an axis defined by the outward-facing surface normal of another surface.

Moving Bodies Relative to Other Geometric Entities

It is also possible to move bodies relative to other geometric entities in the model. The following command takes as arguments two geometric entities. The first entity is the one to move. The second entity is where it will be moved. In both cases, the midpoints of the specified entity are used to determine the distance and direction of the move. "Except" is used to preserve the x, y, or z plane in which the center of the entity lies.

Move {vertex|curve|surface|volume|body} <id_range> [Midpoint] location {vertex|curve|surface|volume|body} <id> [Midpoint] [except [x] [y] [z]]

Moving Merged Entities

The only way that merged entities can be moved is by including each of the merged bodies in the entity list. The following form of the command should be used to move merged entities.

Body <id_list> Move . . . options

All merged entities must be explicitly specified. Any of the move options described above can be used with merged entities.

Move Undo

The Undo option allows a user to reverse the most recent move. The syntax is:

Move Undo

Scale Command

The **scale** commands resizes an entity (body, volume, surface, or curve) by a scaling factor. The scaling factor may be a constant, or may differ in the x, y, and z directions. The entity chosen will be scaled about its centroid, and any mesh on the object will be scaled too, unless the nomesh keyword is used. If the entity chosen belongs to a higher-order entity, for example, you choose to scale a surface that belongs to a body, then the higher-order entity will be scaled.

The command to scale entities is:

{Body|Volume|Surface|Curve} <id_range> [Copy [nomesh]]
Scale {<scale> | x <val> y <val> z <val>}

If the **copy** option is specified, a copy of the entity is made and scaled the specified amount.

Rotate Command

The rotate command rotates a body about a given axis without adding any new geometry. If the Angle or any Components are not specified they are defaulted to be zero. The commands to rotate a body or bodies are:

Body <range> [copy] rotate <angle> about {x | y | z}

Body <range> [copy] rotate <angle> about <x-comp> <y-comp> <z-comp>

Rotate {body|volume|surface|curve|vertex|group} <id_range> about {x|y|z|<xval> <yval> <zval>} angle <val>

Rotate {body|volume|surface|curve|vertex|group} <id_range> about vertex <id> vertex <id> angle <val>

Rotate {body|volume|surface|curve|vertex|group} <id_range> about normal of surface <id> angle <val>

If the copy option is specified, a copy is made and rotated the specified amount.

Reflect Command

The reflect command mirrors the body about a plane normal to the vector supplied. The reflect command will destroy the existing body and replace it with the new reflected body, unless the copy option is used.

Body <range> [copy] reflect <x-comp> <y-comp> <z-comp>

Body <range> [copy] reflect {x | y | z}

Geometry Booleans

- Intersect
- Subtract
- Unite

CUBIT supports booleans operations of intersect, subtract, and unite for bodies.

An automatic function associated with webcutting operations is regularizing geometry which can be turned off or back on with the following command:

set boolean regularize [ON | off]

Intersect

The intersect command generates a new body composed of the space that is shared by the two bodies being intersected. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. The command is:

Intersect <body1_id> with <body2_id> [keep]

The **keep** option results in the original bodies used in the intersect being kept.

Subtract

The subtract operation subtracts one body or set of bodies from another body or set of bodies. The order of subtraction is significant - the body or bodies specified before the **From** keyword is/are subtracted from bodies specified after **From**. The new body retains the original body's id. If any additional bodies are created, they will be given the next highest available ids. The **keep** option simply retains all of the original bodies. The command is:

Subtract [Body] <range> from [Body] <range> [keep]

Unite

The unite operation combines two or more bodies into a single body. The original bodies are deleted and the new body is given the next highest body ID available, unless the **keep** option is used. The commands are:

Unite <body1_id> with <body2_id> [keep]

Unite Body {<range> | all} [keep]

The second form of the command unites multiple bodies in a single operation. If the all option is used, all bodies in the model are united into a single body. If the bodies that are united do not overlap or touch, the two bodies are combined into a single body with multiple volumes.

Geometry Cleanup and Defeaturing

Frequently, models imported from various CAD platforms either provide too much detail for mesh generation and analysis, or the geometric representation is deficient. These deficiencies can often be overcome with small changes to the model. Several tools are provided in CUBIT for this purpose.

The following describes the features available in CUBIT for clean up and defeaturing

- Healing
- Tweaking Geometry
- Removing Geometric Features
- Regularizing Geometry
- Finding Surface Overlap
- Validating Geometry
- Debugging Geometry
- Geometry Accuracy

Healing

Healing is an optional module that detects and fixes ACIS models.

It is possible to create ACIS models that are not accurate enough for ACIS to process. This most often happens when geometry is created in some other modeling system and translated into an ACIS model. Such models may be imprecise due to the inherent numerical limitations of their parent systems, or due to limitations of data transfer through neutral file formats. This imprecision can also result when an ACIS model is created at a different tolerance from the current tolerance settings. This imprecision leads to problems such as geometric errors in entities, gaps between entities, and the absence of connectivity information (topology). Since ACIS is a high precision modeler, it expects all entities to satisfy stringent data integrity checks for the proper functioning of its algorithms. Therefore, if such imprecise models must be processed by an ACIS based system, "healing" of such models is necessary to establish the desired precision and accuracy.

The following sections describe how to use the Healing capability in ACIS and CUBIT to analyze and heal defective ACIS geometry.

- Analyzing Geometry
- Healing Attributes
- Auto Healing
- Spline Removal
- What if Healing is Unsuccessful?

Analyzing Geometry

The following command analyzes the ACIS geometry and will indicate problems detected:

Healer Analyze Body <id_range> [logfile ['filename'] [display]]

The **logfile** option writes the analysis results to the filename specified, or to 'healanalysis.log' by default. In the GUI version of CUBIT, the **display** option will write the results in a dialog window.

The outputs include an estimate of the percentage of good geometry in each body. The optional logfile will include detailed information about the geometry analysis. By default CUBIT will also highlight the bad geometry in the graphics and give a printed summary indicating which entities are "bad". Sample output from this command is shown below:

Percentage good geometry in Body 9: 98%

```
HEALER ANALYSIS SUMMARY:
------------------------
Analyzed 1 Body: 9
Found 2 bad Vertices: 51, 52
Found 3 bad Curves: 76, 77, 80
Found 2 bad CoEdges. The Curves are: 76
Found 1 Bodies with problems: 9
Journaled Command: healer analyze body 9
```

Note that it is not necessary to analyze the geometry before healing; however, it can be useful to analyze first rather than healing unnecessarily. Also note that healer analysis can take a bit of time, depending on the complexity of the geometry and how bad the geometry is.

The validate geometry commands work independently of the healer and give more detailed information.

Healer Settings

You can control the outputs from the healer with the following commands:

Healer Set OnShow {highlight|draw|none}

Healer Set OnShow {badvertices|badcurves|badcoedges|badbodies|all} {On|Off}

Healer Set OnShow Summary {On|Off}

These settings allow you to highlight, draw or ignore the bad entities in the graphics. You can control which entity types to display, as well as whether or not to show the printed summary at the end of analysis.

After you have analyzed the geometry (which can take some time), you can show the bad geometry again with the "**show**" command. This command simply uses cached data (healing attributes - see the next section) from the previous analysis.

Healer Show Body <id_list>

Healing Attributes

Once the geometry is analyzed, the results are stored as attributes on the solid model - this allows you to use the "**show**" command to quickly display the bad geometry again. The results attributes are automatically removed when the geometry is exported or any boolean operations are performed. They can also be explicitly removed with the command

Healer CleanAtt Body <id_range>

You can force the results to be removed immediately after each analyze operation with the "**CleanAtt**" setting (this can save a little memory):

Healer Set CleanAtt {On|Off}

Auto Healing

Healing is an extremely complex process. The general steps to healing are:

- **Preprocess** - trim overhanging surfaces and clean topology (remove small curves and surfaces).
- **Simplify** - converts splines to analytic representations, if possible.
- **Stitch** - geometry cleanup and stitching loose surfaces together to form bodies.
- **Geometry Build** - repairing and building geometry to correct gaps in the model.
- **Post-Process** - calculating pcurves and further repairing bad geometry.
- **Make Tolerant Curves & Vertices** - a last optional step that allows special handling of unhealed entities for booleans - allowing inaccurate geometry to be tolerated.

Autohealing makes these steps automatic with the following command:

Healer Autoheal Body <id_range> [rebuild] [keep] [maketolerant] [logfile ['logfilename'] [display]]

The **rebuild** option unhooks each surface, heals it individually, then stitches all the surfaces back together and heals again. In some cases this can more effectively fix up the body, although it is much more computationally intensive and is not recommended unless normal healing is unsuccessful.

The **keep** option will retain the original body, putting the resulting healed body in a new body.

The **maketolerant** option will make the edges *tolerant* if ACIS is unable to heal them. This can result in successful booleans even if the body cannot be fully healed - ACIS can then sometimes "tolerate" the bad geometry. Note that the healer analyze command will still show these curves as "bad", even though they are tolerant. The validate *geometry* commands however take this into consideration.

The output from the autoheal command can be written to a file using the **logfile** option; the default file name is autoheal.log. The **display** option works as before, displaying the results in a window in the GUI version of CUBIT.

Spline Removal

If healing fails to convert spline surfaces to analytic ones fails, the simplification tolerance can be modified and healing re-run:

Healer Default Simplifytol .1

Healer Autoheal Body 1

Spline surfaces can also be forced into an analytic form (use this command with caution):

Healer Force {plane | cylinder | cone | sphere | torus} Surface <id_list> [Keep]

The Keep option will retain the original body and generate a new body containing analytic surfaces. Note: Spline curves can be found using entity filters:

Execute Filter Curve Geometry_type Spline

What if Healing is Unsuccessful?

The ACIS healing module is under continued development and is improving with every release. However, there will often be situations where healing is unable to fully correct the geometry. This might be okay, as meshing is rarely affected by the small inaccuracies healing addresses. However, boolean operations on the geometry can fail if the bad geometry must be processed by the operation (i.e., a webcut must cut through a bad curve or vertex).

Here are some possible methods to fix this bad geometry:

- Return to the source of the geometry (i.e., Pro/ENGINEER) and increase the accuracy. Re-export the geometry.
- Heal again using the **rebuild** option.
- Heal again using the **make tolerant** option.
- Remove the offending surface from the body (using the remove surface command), then construct new surfaces from existing curves and combine the body back together.
- Composite the surfaces over the bad area, mesh and create a net surface from the composite, remove the bad surfaces and combine.
- Export the geometry as IGES, import the IGES file into a new model and look for double surfaces or surfaces that show up at odd angles using the find overlap commands. Delete and recreate surfaces as needed and combine the surfaces back together into a body.

Contact the development team (cubit-dev@sandia.gov) if you need further help with fixing bad geometry.

Regularizing Geometry

The regularize command removes unnecessary topology, which in effect reverses the imprint operation. This can help clean up the model from extra features that are unnecessary for the geometric definition of the model. The following command regularizes the model:

Regularize Body|Group|Surface|Curve|Vertex <range>

If you are frequently using web-cutting or other boolean operations to decompose your geometry, it may be convenient to always generate regularized geometry. To set creation of reglarized geometry during boolean operations use the following command:

set boolean regularize [ON | off]

Finding Surface Overlap

The surface overlap capability finds surfaces that *overlap* each other, with the capability to specify a distance and angle range between them. This is useful for debugging geometry imprinting and merging problems, as well as for finding gaps in large assembly models. Finding overlapping geometry is done using the command:

Find [Surface] Overlap [{Body|Surface|Volume}<id_list>

]If a list of entities is not specified, all bodies in the model are checked. By default the command does not check the surfaces within a given body against each other; rather, it only checks surfaces between bodies. This can be overridden by inputting a surface list (i.e. **find overlap surface all**), or with a setting (see below).

Facetted Representation

This command works entirely off of the facetted surface representation of the model (the facetted representation is what you see in a shaded view in the graphics). There are inherent advantages and disadvantages with this method. The biggest advantage is avoidance of closest-point calculations with NURBS based geometry, which tends to be slow. This method also eliminates possible problems with unhealed ACIS geometry. The disadvantage is working with a less accurate (i.e., facetted) representation of the geometry. To circumvent problems with this facetted geometry, various settings can be used to control the algorithm. For example, you might consider using a more accurate facetted representation of the model - see below.

Find Overlap Settings

Various settings are used to control the precision and handling of overlaps during the find overlap process. A listing of the settings that find overlap uses is printed using the command:

Find [Surface] Overlap Settings

These settings, and the commands used to control them, are described below.

Facet - Absolute/Angle - The angular tolerance indicates the maximum angle between normals of adjacent surface facets. The default angular tolerance is 15° - consider using a value of 5° . This will generate a more accurate facetted representation of the geometry for overlap detection. This can be particularly useful if the overlap command is not finding surface pairs as you would expect, particularly in "curvy" regions. Note however that the algorithm will run slower with more facets. The distance tolerance means the maximum actual distance between the generated facets and the surface. This value is by default ignored by the facetter - consider specifying a reasonable value here for more accurate results.

set Overlap [Facet] {Angle|Absolute} <value>

Gap - Minimum/Maximum - the algorithm will search for surfaces that are within a distance from the minimum to maximum specified. The default range is 0 to 0.01. Testing has shown this to be about right when searching for coincident surfaces. Gaps can be found by using a range such as 3.95 to 5.05.

set Overlap {Minimum|Maximum} Gap <value>

Angle - Minimum/Maximum - the algorithm will search for surfaces that are within this angle range of each other. The default range is 0.0 to 5.0 degrees. Testing has shown that this range works well for most models. It is usually necessary to have a range up to 5.0 degrees even if you are looking for coincident surfaces because of the different types of faceting that can occur on curvy type surfaces. For example, for the case of a shaft in a hole, the facets of the shaft usually won't be coincident with the facets of the hole, but may be offset by a certain distance circumferentially with each other. The 5 degree max angle range will account for this. If you find that the algorithm is not finding coincident surfaces when it should, you can increase the upper range of this value. Note that this parameter is useful also for finding plates coming together at an angle.

set Overlap {Minimum|Maximum} Angle <value>

Normal - this setting determines whether to search for surfaces whose normals point in the same direction as each other (**same**), away from each other (**opposite**) or either (**any**). The default is ANY, but it may be useful to limit this search to *opposite*, as this would be the usual case for most finds.

set Overlap Normal {ANY|opposite|same}

Tolerance - two individual facets must overlap by more than this area for a match to be found. Consider the two cylindrical curves at the interface of the shaft and the block in Figure 1. Note that some of the facets actually overlap, even though the curves will analytically be coincident. You can filter out false matches by increasing the overlap tolerance area. The default value for this setting is 0.001.

set Overlap Tolerance <value>

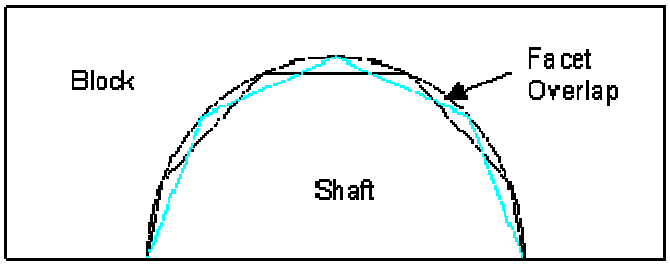


Figure 1. Possible false find due to overlap (tolerance will prevent finding match)

Group - the surface pairs found can optionally be placed into a group. The name of the group defaults to "overlap_surfaces".

set Overlap Group {on|OFF}

List - by default the command lists out each overlapping pair - this can be turned off using the command:

set Overlap List {ON|off}

Display - by default the command clears the graphics and displays each overlapping pair - this can be turned off using the command:

set Overlap Display {ON|off}

Body - by default the command will not search for overlapping pairs within bodies - only between different bodies. Turn this setting on to search for pairs within bodies. Note however that this will slow the algorithm down.

set Overlap Body {on|OFF}

Imprint - If on, Cubit will imprint the overlapping surfaces that it finds together. This will often force imprints that just imprinting bodies together will miss. For each pair of overlapping surfaces, the containing body of one surface is imprinted with the individual curves of the other surface, until the resulting surfaces no longer overlap.

set Imprint {on|OFF}

Validating Geometry

Detailed checks of geometry and topology can be performed using the validate command:

Validate {Body|Volume|Surface|Curve|Vertex|Group} <id_range>

Validate {Volume|Surface|Curve|Vertex} <range> mesh

The **Validate {...} mesh** command performs a connectivity check of the mesh elements to determine the validity of the mesh.

More rigorous checking can be accomplished with the validate geometry commands by specifying a higher check level. Use the following command to accomplish this:

set AcisOption Integer 'check_level' <integer>

where **integer** is one of the following:

10 = Fast error checks

20 = Level 10 checks plus slower error checks (default)

30 = Level 20 checks plus D-Cubed curve and surface checks

40 = Level 30 checks plus fast warning checks

50 = Level 40 checks plus slower warning checks

60 = Level 50 checks plus slow edge convexity change point checks

70 = Level 60 checks plus face/face intersection checks

You can also get more detailed output from the validate command with (the default is *off*):

set AcisOption Integer 'check_output' on

Note that some of the ids listed in the output of the validate command are currently meaningless, e.g. those for coedges.

The validate command can also check for consistent surface normals and return a list of offending surfaces. The syntax for the command is as follows:

Validate [Body] <body_id> Normal [Reference [Surface] <surface_id>] [Reverse]

Using the "reference" keyword, a reference surface is compared to the normal consistency of all other specified surfaces. Inconsistent surfaces can be reversed using the "reverse" keyword.

Debugging Geometry

The following command checks for inconsistencies in the CUBIT topological model, by checking the specified entities and all child topology and/or comparing to solid model topology:

Geomdebug Validate [compare] <entity_list>

This command checks for:

- Consistent CoFace senses
- Loops are closed/complete
- Consistent CoEdge senses
- Correct vertex order on curves w.r.t. parameterization
- Correct tangent direction of curves w.r.t. parameterization

Related Commands:

Geomdebug Vertex <vertex_id>

Geomdebug Curve <curve_id>

Geomdebug Surface <surface_id>

Geomdebug body <body_id>

Geomdebug Containment {Curve | Surface} <id> {Location (options) | Node <id_list>}

The following command prints info about GeometryEntities owned by specified entity:

Geomdebug Geometry <entity_list> [interval <n>] [index <n>] [TEXT] [GRAPHIC] [attributes]

The following command lists (TopologyBridge) topology for specified entity:

Geomdebug solidmodel <entitiy_list> [index <n>]

The following command lists GroupingEntities.

Geomdebug GPE <entity_list>

Geometry Accuracy

The accuracy setting of the ACIS solid model geometry can be controlled using the following command:

[set] Geometry Accuracy <value = 1e-6>

Some operations like imprinting can be more successful with a lower accuracy setting (i.e., 0.1 to 1e-5). However, it is not recommended to change this value. ***Be sure to set it back to 1e-6 before exporting the model or doing other operations as a higher setting can corrupt your geometry***.

Tweaking Geometry

- Tweaking Vertices
- Tweaking Curves
- Tweaking Surfaces

The tweaking commands modify models by moving, offsetting or replacing surfaces or curves, while extending the adjoining surfaces to fill the resulting gaps. This is useful for eliminating gaps between components, simplifying geometry or changing the dimensions of an object.

Tweaking Vertices

The Tweak Vertex command can be used to do the following:

- Tweaking a Vertex With a Chamfer
- Tweaking a Vertex With a Non-Equal Chamfer
- Tweaking a Vertex With a Fillet Radius

Tweaking a Vertex With a Chamfer

Tweak Vertex <id_range> Chamfer Radius <value>

This form of the command creates a chamfered corner at the specified vertex. Can be use on volumes or free surfaces. The 'keep' option creates another volume on which the tweak is applied; the original volume remains unmodified.

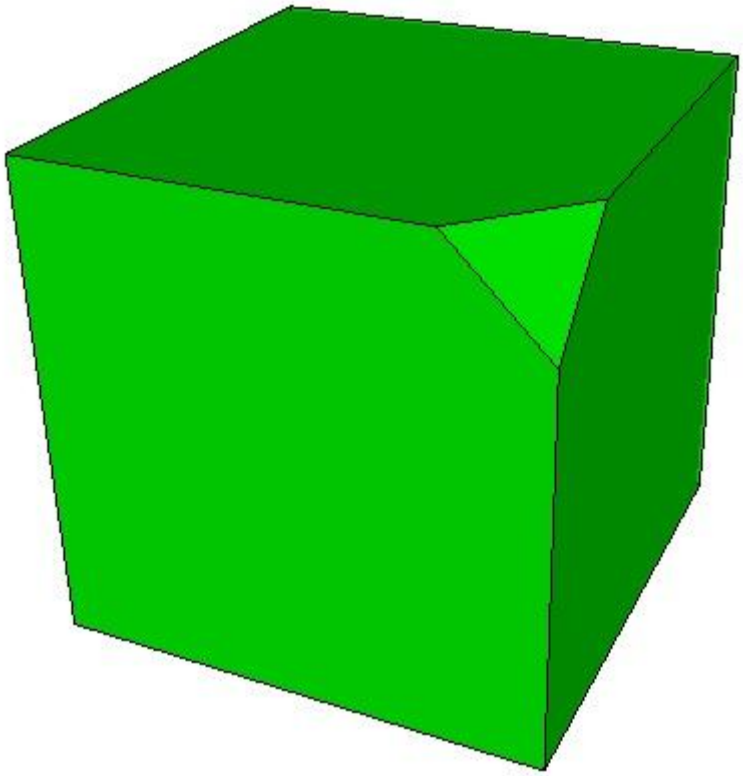


Figure 1. Tweak Vertex Chamfer

Tweaking a Vertex With a Non-Equal Chamfer

Tweak Vertex <id_range> Chamfer Radius <value> [Curve <id> Radius <value> Curve <id> Radius <value> Curve <id>] [keep]

This next form of the command creates a non-equal chamfered corner at the specified vertex. Can only be used on vertices of volumes. The 'keep' option creates another volume on which the tweak is applied; the original volume remains unmodified.

Tweaking a Vertex With a Fillet Radius

Tweak Vertex <id_range> Fillet Radius <value> [keep]

This command replaces a vertex with a filleted radius. The command can only be used on free surfaces. The 'keep' option creates another volume on which the tweak is applied; the original free surface remains unmodified.

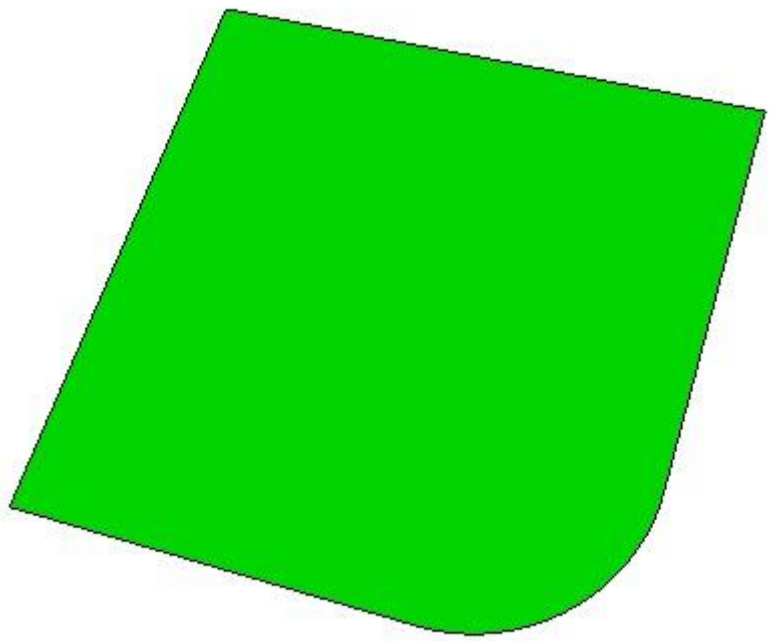


Figure 2. Tweak Vertex Fillet

Tweaking Curves

The Tweak Curve command is used to modify a curve via filleting, and creating a new blended interface between the two surfaces which share this edge. This interface may be chamfered or blended. The radius value is the radius of the fillet arc created. The command syntax is:

Tweak Curve <id_range> {Blend|Chamfer} Radius <value>

Figure 1 shows an isometric brick with a blended edge. Figure 2 shows an identical brick with a chamfered edge.

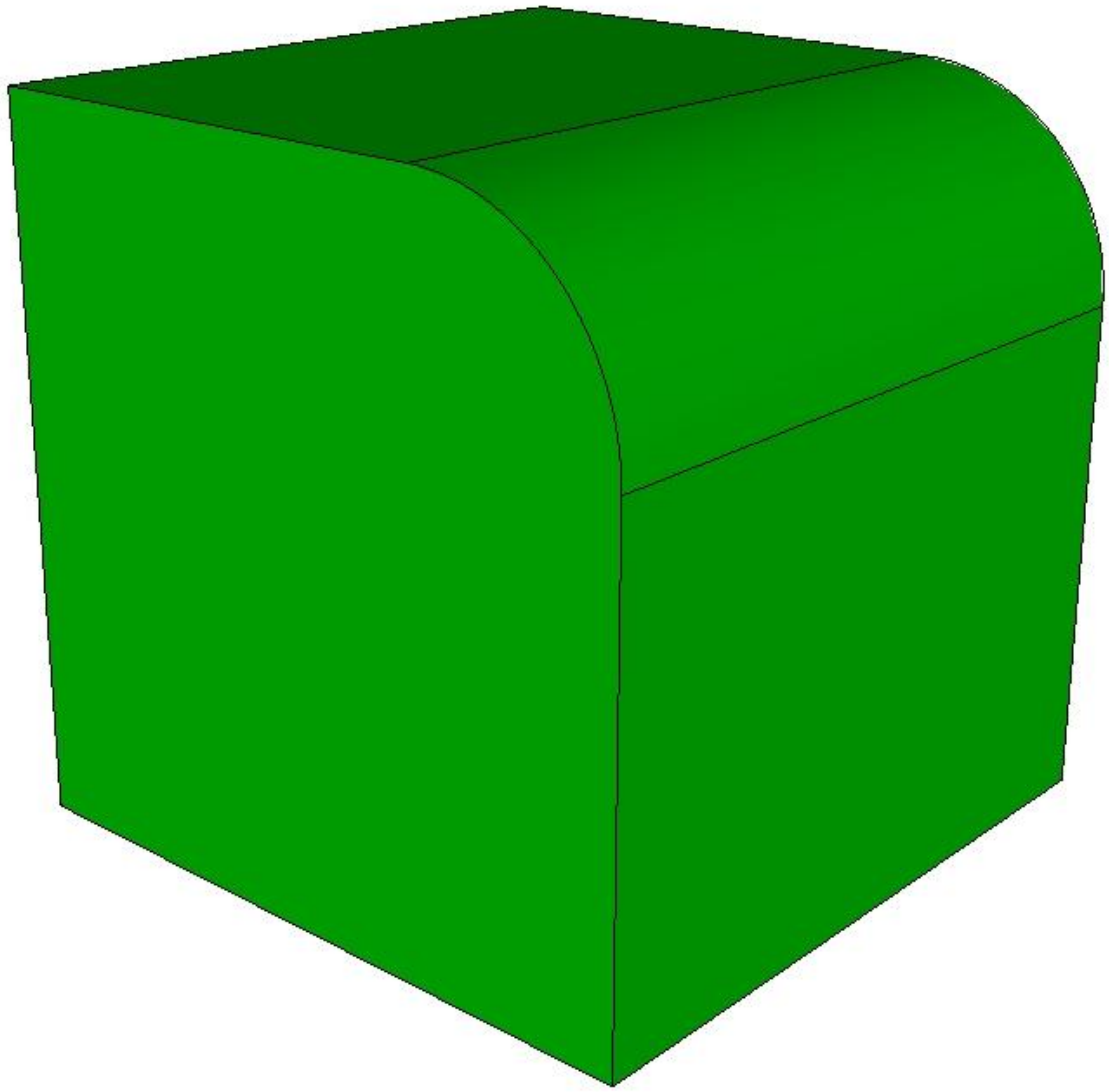


Figure 1. Tweak Via Edge Blending

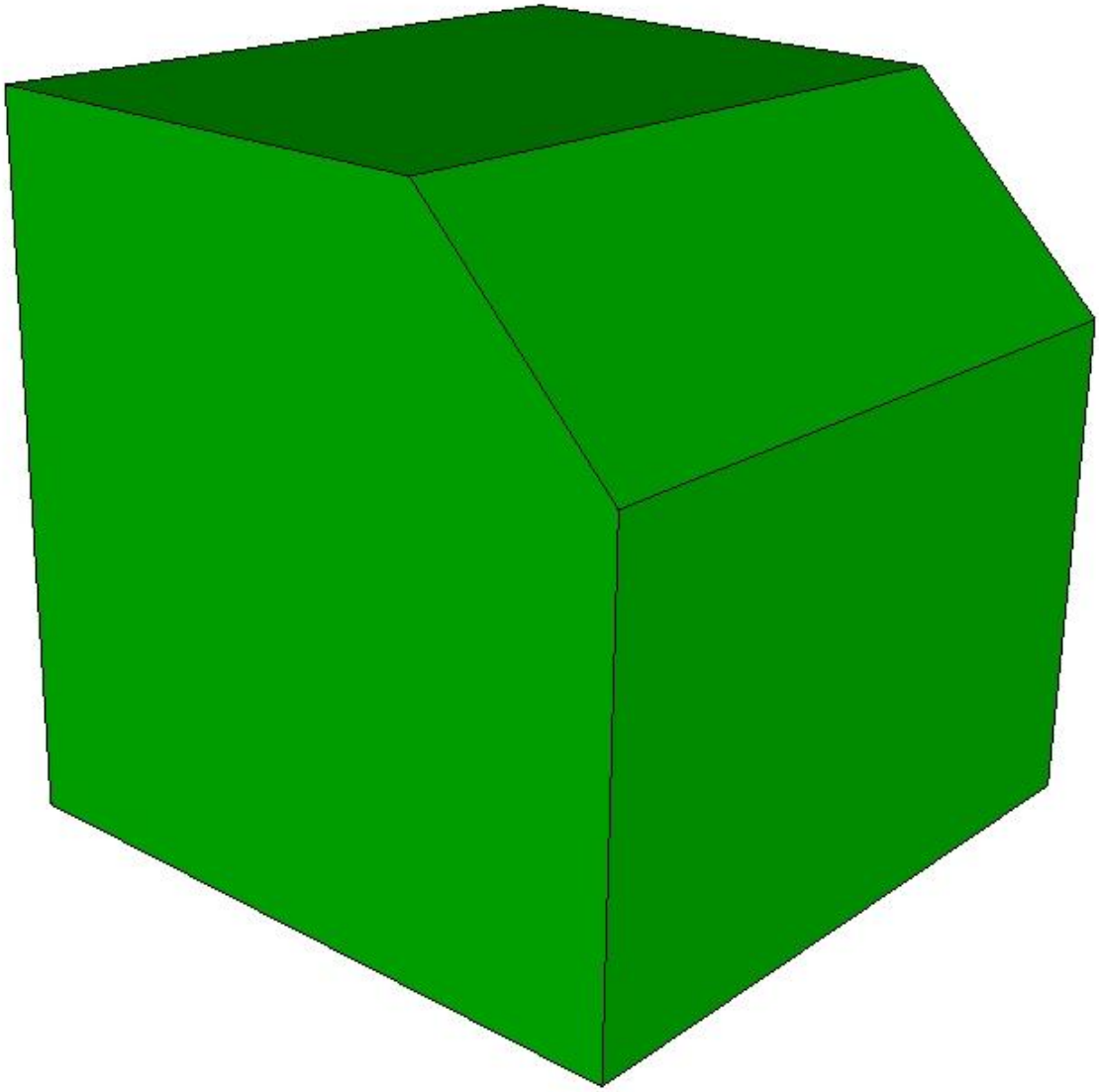


Figure 2. Tweak Via Edge Chamfering

Tweaking Surfaces

The following options of the Tweak Surface command are available. Command syntax and examples follow below.

- Tweaking a Surface Using an Offset
- Tweaking a Surface by Moving
- Tweaking a Surface by Replacement
- Tweaking a Conical Surface

Tweaking a Surface Using an Offset

Tweak Surface <id_range> Offset <value> [keep]

The first form offsets an existing set of surfaces and extends the attached surfaces to meet them. A positive offset value will offset the surface in the positive surface normal direction while a negative value will go the other way. Figure 1 shows a simple example of offsetting. Note that you can also offset whole groups of surfaces at once. The keep option will retain the original surfaces and curves.

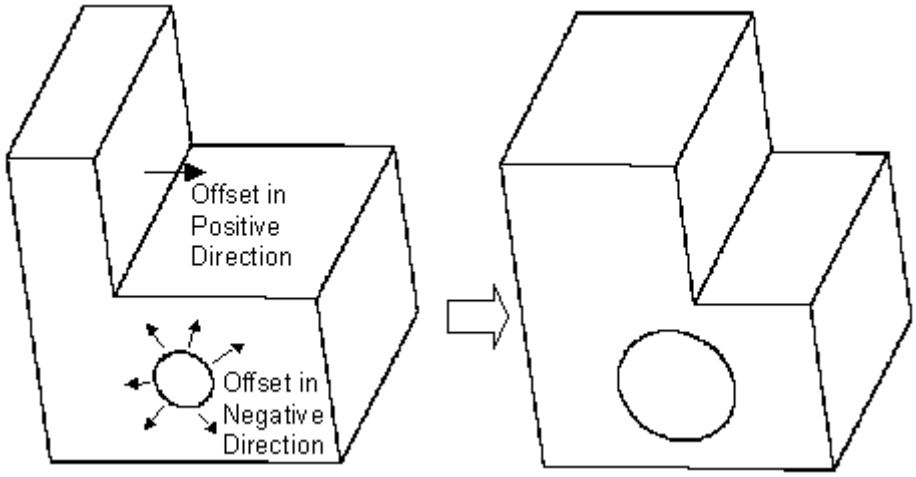


Figure 1. Tweak Offset

Tweaking a Surface by Moving

The move forms of the command simply move the given surfaces along a vector direction. The direction can be specified either absolutely or relative to other geometry entities in the model (from entity centroid to location). Note that when moving a surface for tweak, the surface is moved and it and the adjoining surfaces are extended or trimmed to match up again. So, for example, moving a vertically oriented planar surface in the vertical direction will have no effect. In this example, if you move the surface 10 in the x and 5 in the y the effect will be to move it simply 10 in the x. You can also use this form of the command to move a protrusion around - just be sure to specify all of the surfaces on the protrusion for moving. The last form of the command can be used to move a surface along another surface's normal.

Tweak Surface <id_range> Move {Vertex|Curve|Surface|Volume|Body} <id> Location {Vertex|Curve|Surface|Volume|Body} <id> [Except [X][Y][Z]] [keep]

Tweak Surface <id_range> Move {Vertex|Curve|Surface|Volume|Body} <id> Location <x_val> <y_val> <z_val> [Except [X][Y][Z]] [keep]

Tweak Surface <id_range> Move <dx_val> <dy_val> <dz_val> [keep]

Tweak Surface <id_range> Move Normal To Surface <id> Distance <val> [Except [X][Y][Z]] [keep]

Tweaking a Surface by Replacement

The 'replace' form of the command actually replaces the given surface with a copy of the new surface, then extends and trims surfaces to match up. This can be useful for closing gaps between components or performing more complicated modifications to models. The command syntax is:

Tweak Surface <id_range> Replace [With] Surface <id> [keep] [reverse]

Figure 2 shows a simple example.

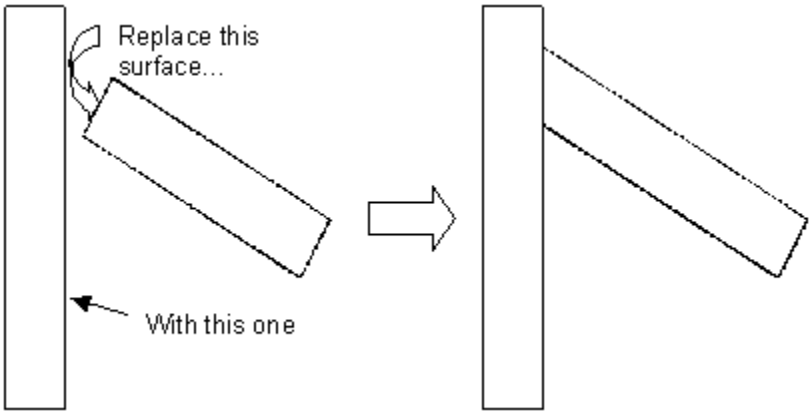


Figure 2. Tweak Replace Surface (Viewed directly from the side)

Tweaking a Conical Surface

The last form of the command is used to replace a conical projection with a flat circular surface. This command is useful for simplifying bolt holes. The command syntax is.

Tweak Surface <id_range> Cone

The following is a simple example illustrating the use of the tweak surface cone command.

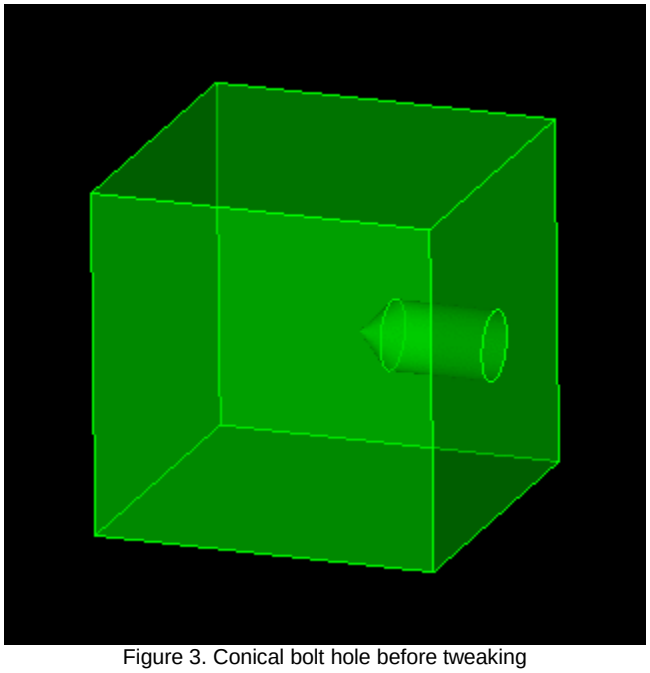


Figure 3. Conical bolt hole before tweaking

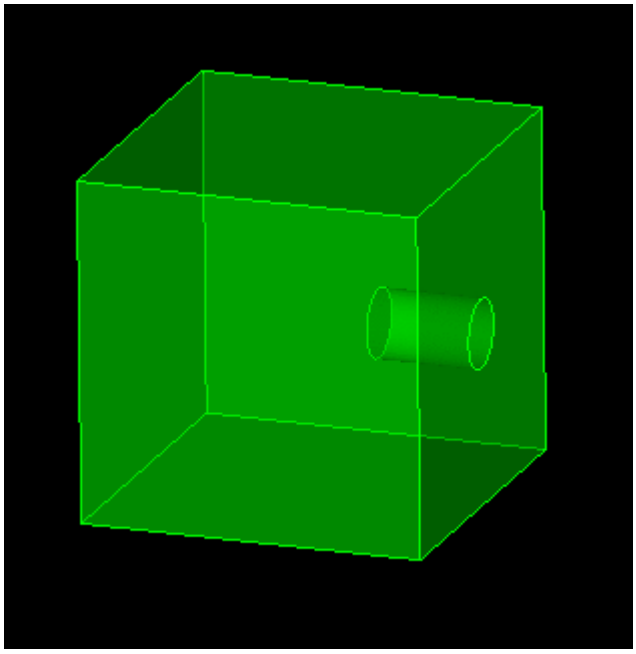


Figure 4. Bolt hole after tweaking

Removing Geometric Features

- Vertex Removal
- Surface Removal

The Remove will remove surfaces or vertices from bodies. Adjacent surfaces or curves will be extended, where possible, to fill in remaining gaps. The remove command is useful for replacing filleted edges with sharp corners.

Removing Vertices

At times you may find that you have an extraneous vertex in your model. This would be a vertex connected to two and only two edges. This stray vertex can cause unwanted mesh artifacts, due to the fact that a mesh node MUST lie on this vertex, thereby disallowing the possibility of movement for better quality. Fortunately there is a relatively easy way of getting rid of this stray vertex using the tweak surface command.

Tweak Surface <id> replace with Surface <same_id>

Note that you are replacing a surface with itself. In doing so, the geometry engine will do an intersection check on that surface, and should realize that the vertex doesn't need to be there.

Removing Surfaces

The remove surface command removes surfaces from bodies. By default, it attempts to extend the adjoining surfaces to fill the resultant gap. This is a useful way to remove fillets and rounds and other features such as bosses not needed for analysis. See Figure 1 for an example of this process. The syntax for this command is:

Remove Surface <id_range> [EXTEND|noextend] [keepsurface] [keep]

The **noextend** qualifier prevents the adjoining surfaces from being extended, leaving a gap in the body. This is sometimes useful for repairing bad geometry - the surface can be rebuilt with surface from curves or a net surface, etc.., then combined back onto the body.

The **keep** option will retain the original body and put the results of the remove surface in a new body.The **keepsurface** option will retain the surface which was removed.

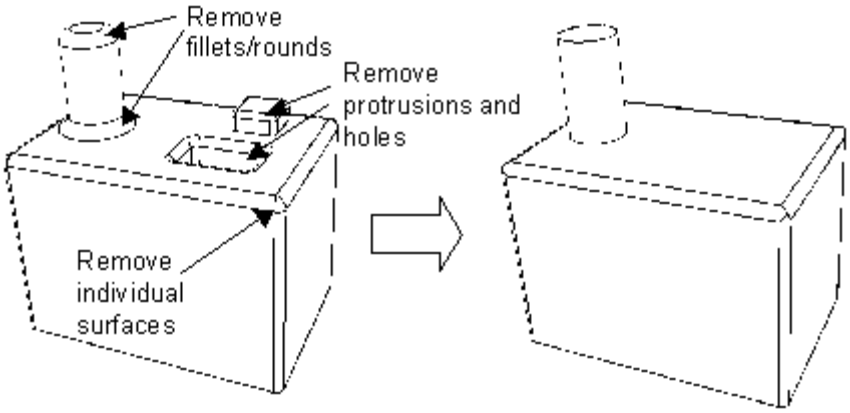


Figure 1. Remove Surface Example

Remove Sliver Surface

This command uses the ACIS remove surface capability on surfaces that have area less than a specified area limit. When ACIS removes a surface it extends the adjoining surfaces and reintersects them to fill the gap. If it is not possible to extend the surfaces or if the geometry is bad the command will fail. The syntax for this command is:

> **Remove Slivers Body <id_range> [EXTEND|noextend] [keepsurface] [keep] [arealimit [<double>]] Default arealimit = 0.1**

The **noextend**, **keepsurface** and **keep** options operate as for the remove surface command. The **arealimit** option allows the user to set the area below which surfaces will be removed.

Trimming and Extending Curves

Curves can be trimmed or extended with the following command:

> Trim Curve <id> AtIntersection {Curve|Vertex <id>} Keepside Vertex <id> [near]

Curves can be trimmed or extended where they intersect with another curve or at a vertex location. When trimming to another curve, the curves must physically intersect unless they both are straight lines in which case the **near** option is available. With the **near** option the closest intersection point is used to the other line - so it is possible to trim to a curve that lies in a different plane. When trimming to a vertex, if the vertex does not lie on the curve, it is projected to the closest location on the curve or an extension of the curve if possible.

The **Keepside** vertex is needed to determine which side of the curve to keep and which side to throw away. This vertex need not be one of the curve's vertices, nor does it need to lie on the curve. However, if it is not on the curve it will be projected to the curve and that location will determine which side of the curve to keep.

If the curve is part of a body or surface, it is simply copied first before trimming/extending. If it is a free curve a new curve is created and the old curve is removed. The figures below show several examples of trimming/extending curves.

Trimming a Curve

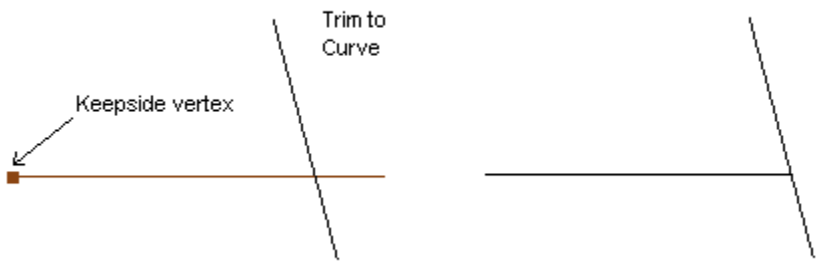


Figure 1. Trimming a Curve to an Intersecting Curve

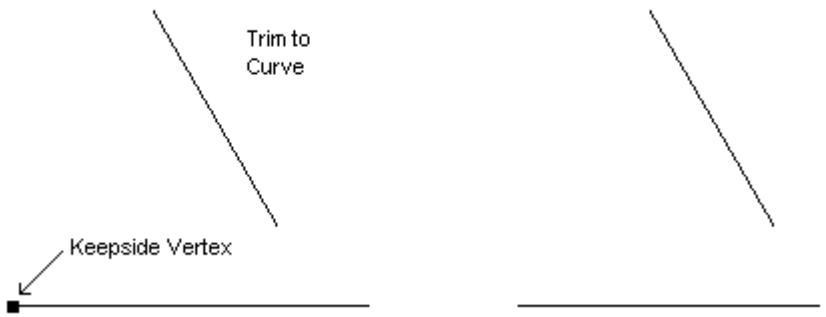


Figure 2. Trimming a Curve to a Non-Intersecting Curve Using the Near Option

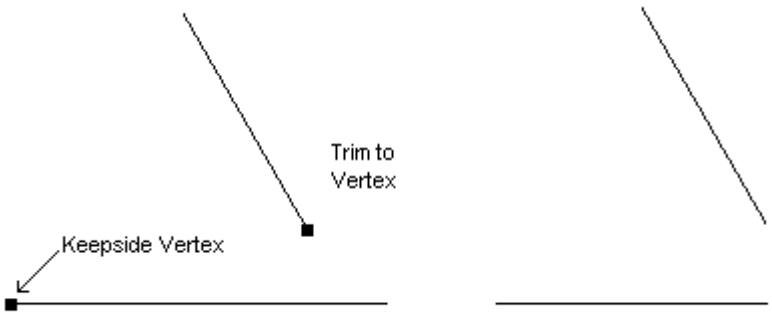


Figure 3. Trimming a Curve to a Vertex

Extending a Curve

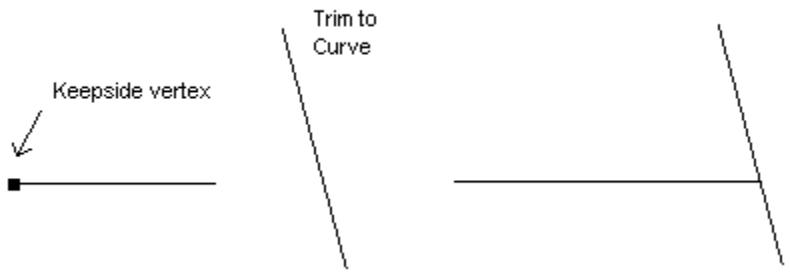


Figure 4. Extending a Curve to An Intersecting Curve

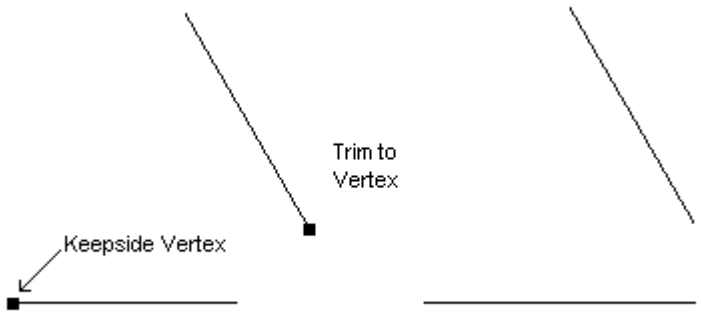


Figure 5. Extending a Curve to a Non-Intersecting Vertex Using the Near Option

Geometry Decomposition

Geometry decomposition is often required to generate an all-hexahedral mesh for three-dimensional solids, as fully automatic all-hex mesh generation of arbitrary solids is not yet possible in CUBIT. While geometry booleans can be used for decomposition (and are the basis of the underlying implementation of advanced decomposition tools described here), CUBIT has a webcut capability specially tuned for decomposition. It is also useful to split periodic surfaces to facilitate quad and hex meshing.

- Web Cutting
- Splitting Geometry
- Section Command
- Separating Multi-Volume Bodies

Web Cutting

The term "web cutting" refers to the act of cutting an existing body or bodies, referred to as the "blank", into two or more pieces through the use of some form of cutting tool, or "tool". The two primary types of cutting tools available in CUBIT are surfaces (either pre-existing surfaces in the model or infinite or semi-infinite surfaces defined for webcutting), or pre-existing bodies.

The various forms of the webcut command can be classified by the type of tool used for cutting. These forms are described below, starting with the simplest type of tool and progressing to more complex types.

- Webcut Using the Chop Command
- Webcut Using Planar or Cylindrical Surface
- Webcut with Arbitrary Surface
- Webcut Using Tool or Sheet Body
- Webcut by Sweeping Curves or Surfaces
- Webcut Options
- Webcut Preview

General Notes

The primary purpose of web cutting is to make an existing model meshable with the hex meshing algorithms available in CUBIT. While web cutting can also be used to build the initial geometric model, the implementation and command interface to web cutting have been designed to serve its primary purpose. Several important things to remember about webcutting are as follows:

- The geometric model should be checked for integrity (using imprinting and merging) before starting the decomposition process. This makes the checking process easier, since there are fewer bodies and surfaces to check. Once the model passes that initial integrity check, it is rare that decompositions using webcut will result in a model that does not also pass the same checks.
- The use of the Imprint option can in cases save execution time, since it limits the scope of the imprint operations and thereby works faster. The alternative is performing and Imprint All on the pieces of the model after all decompositions have been completed; this operation has been made much faster in more current releases of CUBIT, but will still take a noticeable amount of time for complicated models.
- While the Webcut commands make it very simple to cut your model into very many pieces, we recommend that the user restrict the decomposition they perform to only that necessary for meshability or for obtaining an acceptable mesh. Having more volumes in the model may simplify individual volumes, but may not always result in a higher quality mesh; it will always increase the run time and complexity of the meshing task.
- When the webcut command is executed the associated geometry will be regularized. This behavior can be changed, see geometry booleans.

The Appendix and the Power Tools Tutorial contain some examples that demonstrate the use of webcutting operations.

Chop Command

The **chop** command works similarly to a webcut command, but is faster. Given two bodies, the command will find the intersection of the two bodies, and divide the main body into a body that lies outside the intersection, and a body that lies inside the intersection. The tool body will be deleted, unless the keep option is specified. The syntax of the command is:

Chop [body] <body_id> with [body] <body_id> [keep] [nonreg]

The nonreg option results in the bodies being non-regularized.

Web Cutting with a Planar or Cylindrical Surface

The commands used to **webcut** with a planar or cylindrical surface in CUBIT are:

- Coordinate Plane
- Planar Surface
- Plane from 3 Points
- Plane Normal to Curve
- Cylindrical Surface

- Previewing a Web Cut Plane

Coordinate Plane

In the command's simplest form, a coordinate plane can be used to cut the model, and can optionally be offset a positive or negative distance from its position at the origin.

> Webcut {Volume|Body|Group} <id_range> [With] Plane {xplane|yplane|zplane} [Offset <val>] [rotate <theta> about x|y|z <xval> <yval> <zval> [center <xval> <yval> <zval>]] webcut_options [preview]

The cutting plane can be rotated about a user-specified axis using the **rotate** option. The center of rotation can be moved by using the **center** option.

Selecting the preview option allows the user to preview the webcutting plane.

Planar Surface

An existing planar surface can also be used to cut the model; in this case, the surface is identified by its ID as the cutting tool.

> Webcut {Volume|Body|Group} <id_range> [With] Plane Surface <surface_id> webcut_options

The planar surface to be used for web cutting can also be previewed using the Draw Plane command.

Plane from 3 Points

Any arbitrary planar surface can be used by specifying three vertices that define the plane.

> Webcut {Volume|Body|Group} <id_range> [With] Plane Vertex <vertex_1> [Vertex] <vertex_2> [Vertex] <vertex_3> webcut_options

The Draw Plane command will also provide a preview of the infinite plane used for web cutting for this case.

Plane Normal to Curve

The next command allows a user to specify an infinite cutting plane by specifying a location on a curve. The cutting plane is created such that it is normal to the curve tangent at the specified location.

> Webcut {body|group} {<body_id_range>|all} [with] Plane Normal To Curve <curve_id> {fraction <f> | distance <d> | position <xval><yval><zval> | close_to vertex <vertex_id>} [[From] Vertex <vertex_id> (optional for 'fraction' & 'distance')] webcut_options

The position on the curve can be specified as:

1. A fraction along the curve from the start of the curve, or optionally, from a specified vertex on the curve.
2. A distance along the curve from the start of the curve, or optionally, from a specified vertex on the curve.
3. An xyz position that is moved to the closest point on the given curve.
4. The position of a vertex that is moved to the closest point on the given curve.

The point on the curve can be previewed with the Draw Location On Curve command and the plane to be used for the webcut can be previewed with the Draw Plane command.

Cylindrical Surface

Finally, A semi-infinite cylindrical surface can be used by specifying the cylinder radius, and the cylinder axis. The axis is specified as a line corresponding to a coordinate axis, the normal to a specified surface, two arbitrary points, or an arbitrary point and the origin. The "center" point through which the cylinder axis passes can also be specified.

> Webcut {blank} cylinder radius <val.> axis {x|y|z|normal of surface <id>| vertex <id_1> vertex <id_2>| <x_val> <y_val> <z_val>>} [center <x_val> <y_val> <z_val>] webcut_options

Previewing a Web Cut Plane

The ability to preview a plane prior to webcutting or creating the plane is possible with the following commands:

> Draw Plane Vertex <v1_id> [vertex] <v2_id> [vertex] <v3_id> [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']
>
> Draw Plane Surface <surface_id> [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']
>
> Draw Plane {xplane|yplane|zplane} [offset <val>] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']
>
> Draw Plane Normal To Curve <curve_id> {fraction <f> | distance <d> | position <xval><yval><zval> | close_to vertex <vertex_id>} [[from] vertex <vertex_id> (optional for 'fraction' & 'distance')] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

The first passes a plane through 3 vertices, the second uses an existing plane, the third draws a plane normal to one of the global axes, and the fourth draws a plane normal to the tangent of a curve at a location along the curve. By default, the commands draw the plane just large enough to intersect the bounding box of the entire model with minimum surface area. Optionally, you can give a list of bodies to intersect for this calculation. You can also extend the size of the surface by either a percentage distance or an absolute distance of the minimum area size. The default color is blue, but you can specify a different one. See the Appendix of the CUBIT Users Guide for available colors in CUBIT.

The cylinder used for the webcut operation can be previewed using the Draw Cylinder Command.

> Draw Cylinder Radius <val> Axis {x|y|z|Vertex <id_1> Vertex <id_2> | <xyz values>} [Center <x_val> <y_val> <z_val>] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

The cylinder is defined by a radius and the cylinder axis. The axis is specified as a line corresponding to a coordinate axis, the normal to a specified surface, two arbitrary points, or an arbitrary point and the origin. The center point through which the cylinder axis passes can also be specified.

By default, the commands draw the cylinder just large enough to just intersect the bounding box of the entire model. Optionally, you can give a list of bodies to intersect for this calculation. You can also extend the length of the cylinder by either a percentage distance or an absolute distance of the cylinder length. The default color is blue, but you can specify a different one. See the Appendix of the CUBIT Users Guide for available colors in CUBIT.

Web Cutting with an Arbitrary Surface

An arbitrary "sheet" surface can also be used to webcut a body. This sheet need not be planar, and can be bounded or infinite. The following commands are used:

> webcut {blank} with sheet {body|surface} <id> [webcut_options]
>
> webcut {blank} with sheet extended [from] surface <id> [webcut_options]

In its first form, the command uses a sheet body, either one that is pre-existing or one formed from a specified surface. Note that in this latter case the (bounded) surface should completely cut the body into two pieces. Sheet bodies can be formed from a single surface, but can also be the combination of many surfaces; this form of webcut can be used with quite complicated cutting surfaces.

Extended sheet surfaces can also be used; in this case, the specified surface will be extended in all directions possible. Note that some spline surfaces are limited in extent, and so these surfaces may or may not completely cut the blank.

Web Cutting using a Tool or Sheet Body

Any existing body in the geometric model can be used to cut other bodies; the command to do this is:

Webcut {blank} tool [body] <id> [webcut_options]

This simply uses the specified tool body in a set of boolean operations to split the blank into two or more pieces.

Another form of the command cuts the body list with a temporary sheet body formed from the curve loop. This is the same sheet as would be created from the command Create Surface Curve <id_list>.

Webcut {Body|Group} <id_range> [With] Loop [Curve] <id_range> NOIMPRINT|Imprint] [NOMERGE|Merge] [group_results]

Webcut {Volume|Body|Group} <id_range> [With] Bounding Box {Body|Volume|Surface|Curve|Vertex <id_range>} [Tight] [[Extended] {Percentage|Absolute} <val>] [{X|Width} <val>] [{Y|Height} <val>] [{Z|Depth} <val>]] NOIMPRINT|Imprint] [NOMERGE|Merge] [group_results]

The final form of this command webcuts a body with the bounding box of another entity. This bounding box may be tight or extended.

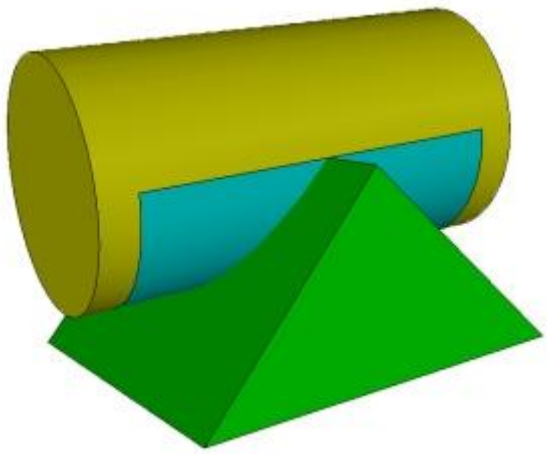


Figure 1. Cylinder webcut with bounding box of prism.

Web Cutting by Sweeping Curves or Surfaces

Webcutting with sweeping creates a swept tool body in the same step as the webcut operation. There are 4 general ways to **webcut** with sweeping:

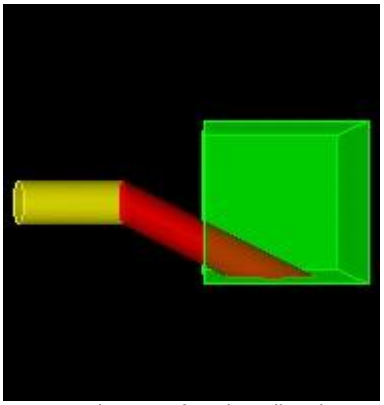
- Webcut by sweeping a surface along a trajectory
- Webcut by sweeping a surface about an axis
- Webcut by sweeping a curve(s) along a trajectory
- Webcut by sweeping a curve(s) about an axis

Webcut by sweeping a surface along a trajectory

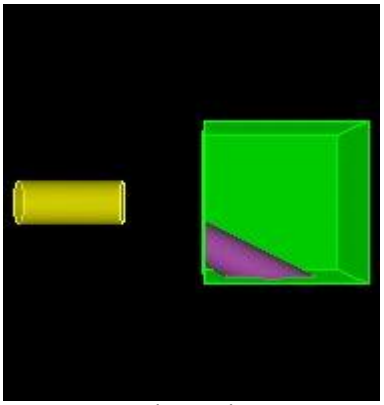
This command allows one or more surfaces to be swept, creating a volume that is used for the webcut. If more than one surface is specified, the surfaces must contain coincident curves. The surfaces are swept along a direction and some distance or perpendicular and some distance or along a curve. For best results the curve to sweep the surface along should intersect one of the surfaces. The **through_all** option will sweep the surfaces along the trajectory far enough so as to intersect all input bodies. The **stop surface <id>** option is used to identify a surface at which the sweep will stop. If using this option when sweeping along a curve, the sweep will stop at the first place possible. The **up_to_next** option indicates that the user wants to webcut with only the first water tight volume that forms as a result of the intersection between sweep and union of all blank bodies. The **up_to_next** option should not used when defining the webcut with a vector and a distance or **perpendicular** and a distance. The **through all** options should not used when defining the webcut with a vector and a distance

Webcut {Volume|Body|Group} <id> sweep surface <id_range> {vector <x> <y> <z> [distance <distance>] | perpendicular distance <distance>} [through_all | stop surface <id> | up_to_next] [webcut_options]

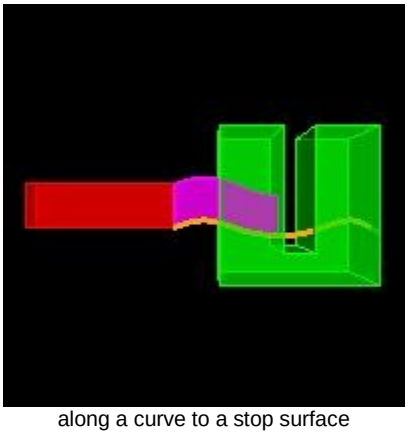
Webcut {Volume|Body|Group} <id> sweep surface <id_range> along curve <id> [through_all | stop surface <id> | up_to_next] [webcut_options]



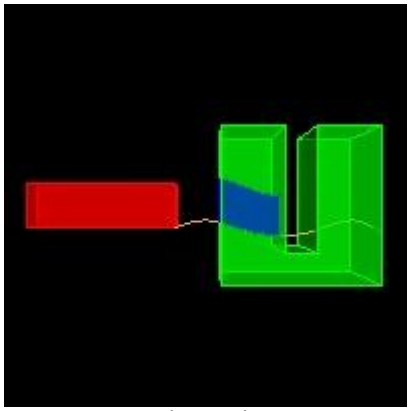
sweeping a surface in a direction



resultant webcut



along a curve to a stop surface



resultant webcut

Figure 1. Examples of web cutting with swept surfaces

Webcut by sweeping a surface about an axis

This command allows a one or more surfaces to be swept, creating a volume that is used for the webcut. If more than one surface is specified, the surfaces must contain coincident curves. The surface is swept about a user-defined axis or about one of the x y z coordinate axes and a specified angle. The **stop surface <id>** option is used to identify a surface at which the sweep will stop. The **up_to_next** option indicates that the user wants to webcut with only the first water tight volume that forms as a result of the intersection between sweep and union of all blank bodies. For these 2 options to work correctly the user must specify an angle large enough for the rotation to traverse the **stop surface** or the **up_to_next** surface.

Webcut {Volume|Body|Group} <id> sweep surface <id_range> {axis <xpoint ypoint zpoint xvector yvector zvector> | xaxis | yaxis | zaxis } angle <degrees> [stop surface <id> | up_to_next] [webcut_options]

Webcut by sweeping a curve(s) along a trajectory

This command allows a curve(s) to be swept, creating a surface that is used for the webcut. If multiple curves are specified, they must share vertices and form a continuous path. The curve(s) is swept along a direction and some distance or along another curve. If sweeping a curve(s) along another curve, for best results the curve(s)-to-swept and the curve to sweep along should intersect at some point. The **stop surface <id>** option is used to identify a surface at which the sweep will stop. If using this option when sweeping along a curve, the sweep will stop at the first place possible. The **through_all** option will sweep the curve(s) along the trajectory far enough so as to intersect all input bodies. For the webcut to be successful, the swept curve(s) must completely traverse a portion of a blank body(s), cutting off a complete piece of the blank body(s). Option **through_all** should not be used when defining the webcut with a vector and a distance or along a curve.

Webcut {Volume|Body|Group} <id> sweep curve <id_range> {vector <x> <y> <z> [distance <distance>| along curve <id>] } [through_all | stop surface <id>] [webcut_options]

Webcut by sweeping a curve(s) about an axis

This command allows a curve to be swept, creating a surface that is used for the webcut. If multiple curves are specified, they must share vertices and form a continuous path. The curve(s) is swept about a user-defined axis or about one of the x y z coordinate axes and a specified angle. For the webcut to be successful, the swept curve(s) must completely traverse a portion of a blank body(s), cutting off a complete piece of the blank body(s). The **stop surface <id>** option is used to identify a surface at which the sweep will stop. For this option to work correctly the user must specify an angle large enough for the rotation to traverse the **stop surface**.

Webcut {Volume|Body|Group} <id> sweep curve <id_range> {axis <xpoint ypoint zpoint xvector yvector zvector> | xaxis | yaxis | zaxis } angle <degrees> [stop surface <id>] [webcut_options]

Web Cutting Options

The following options can be used with all webcut commands:

Group_results: The various pieces resulting from the previous command are placed into a group named `webcut_group'.

[Imprint | Noimprint]: In its default implementation, webcutting results in the pieces not being imprinted on one another; this option forces the code to imprint the pieces after webcutting.

[Merge | Nomerge]: By default, the pieces resulting from an imprint are manifold; specifying this option results in a merge check for all surfaces in the pieces resulting from the webcut.

Web Cutting Preview

Preview a Webcutting Plane

The ability to preview a plane prior to webcutting or creating the plane is possible with the following commands:

Draw Plane Vertex <v1_id> [vertex] <v2_id> [vertex] <v3_id> [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

Draw Plane Surface <surface_id> [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

Draw Plane {xplane|yplane|zplane} [offset <val>] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

Draw Plane Normal To Curve <curve_id> {fraction <f> | distance <d> | position <xval><yval><zval> | close_to vertex <vertex_id>} [[from] vertex <vertex_id> (optional for 'fraction' & 'distance')] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

The first passes a plane through 3 vertices, the second uses an existing plane, the third draws a plane normal to one of the global axes, and the fourth draws a plane normal to the tangent of a curve at a location along the curve. By default, the commands draw the plane just large enough to intersect the bounding box of the entire model with minimum surface area. Optionally, you can give a list of bodies to intersect for this calculation. You can also extend the size of the surface by either a percentage distance or an absolute distance of the minimum area size. The default color is blue, but you can specify a different one. See the Appendix of the CUBIT Users Guide for available colors in CUBIT.

Preview a Web Cutting Cylinder

The ability to preview a cylinder prior to webcutting is possible with the following command:

> Draw Cylinder Radius <val> Axis {x|y|z|Vertex <id_1> Vertex <id_2> | <xyz values>} [Center <x_val> <y_val> <z_val>] [[intersecting] Body <id_range>] [extended percentage|absolute <val>] [color 'color_name']

The cylinder is defined by a radius and the cylinder axis. The axis is specified as a line corresponding to a coordinate axis, the normal to a specified surface, two arbitrary points, or an arbitrary point and the origin. The center point through which the cylinder axis passes can also be specified.

By default, the commands draw the cylinder just large enough to just intersect the bounding box of the entire model. Optionally, you can give a list of bodies to intersect for this calculation. You can also extend the length of the cylinder by either a percentage distance or an absolute distance of the cylinder length. The default color is blue, but you can specify a different one. See the Appendix of the CUBIT Users Guide for available colors in CUBIT.

Splitting Geometry

The Split command divides curves or surfaces into multiple entities. The command results are similar to imprinting. However, vertex and/or curve creation is not necessary for the split command.

- Split Curve
- Split Surface
- Split Periodic Surfaces

Split Curve

The Split Curve command will split a curve without the need for geometry creation (unlike imprinting). The syntax is shown below.

> Split Curve <id> [location on curve options] [Preview]

To split a curve, simply specify a location or a location on curve (see location specification). Using the **Preview** keyword will draw the splitting location on the curve..

Split Surface

The Split Surface command divides one or more surfaces into multiple surfaces. The command results are similar to imprint with curve. However, curve creation is not necessary for splitting surfaces. Two primary forms of the command are available.

- Split Across
- Split (Automatically)

The first form splits a single surface using locations while the second splits either a single surface or a chain of surfaces in an automatic fashion.

Split Across

Two forms of Split Across are available:

Split Surface <id> Across [Pair] Location <options multiple locs> [Preview [Create]]

Split Surface <id> Across Location <multiple locs> Onto Curve <id> [Preview] Create]]

This command splits a surface with a spline projection through multiple locations on the surface. See Location, Direction, and Axis Specification for a detailed description of the location specifier. Figure 1 shows a simple example of splitting a single surface into two surfaces. A temporary spline was created through the three specified locations (Vertex 5 6 7), and this curve was used to split the surface.

split surface 1 across location vertex 5 6 7

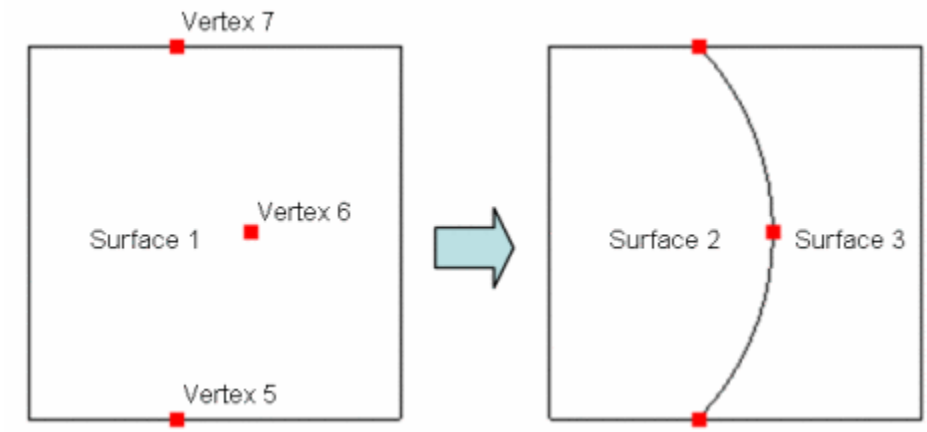


Figure 1 - Splitting Across with Multiple Locations

The **Pair** keyword will pair locations to create multiple surface splitting curves (each defined with two locations). An even number of input locations is required. Figure 2 shows an example:

split surface 1 across pair vertex 5 7 6 8

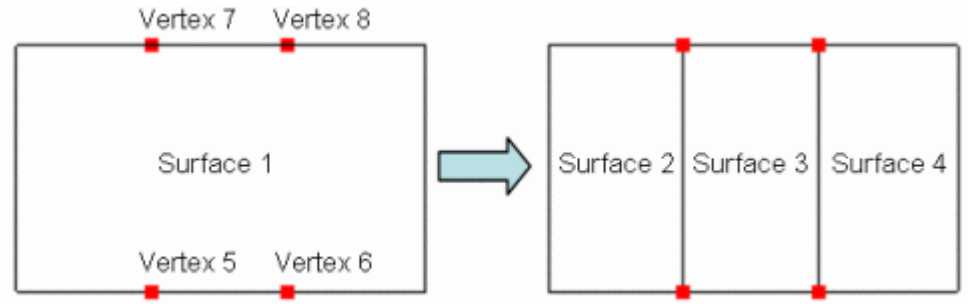


Figure 2 - Splitting Across with Pair Option

The **Preview** keyword will show a graphics preview of the splitting curve. If the **Create** keyword is also specified, a free curve (or curves) will be created - these are the internal curves that are used to imprint the surfaces.

The **Onto Curve** format of the command takes one or more locations on one side of the surface and projects them onto a single curve on the other side of the surface. Figure 3 shows an example:

split surface 1 across vertex 5 6 onto curve 4

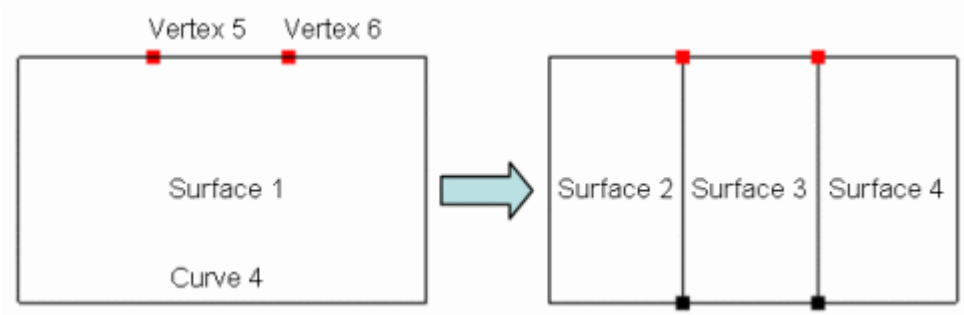


Figure 3 - Splitting Across with Onto Curve

Split (Automatically)

This form of the command splits a single surface or a chain of surfaces in an automatic fashion. It is most convenient for splitting a fillet or set of fillets down the middle - oftentimes necessary to prepare for mesh sweeping.

> Split Surface <id_list> [Corner Vertex <id_list>] [Direction Curve <id>] [Segment|Fraction|Distance <val> [From Curve <id>]] [Through Vertex <id_list>] [Parametric <on|OFF>] [Tolerance <val>] [Preview [Create]]

- Logical Rectangle
- Split Orientation
- Corner Vertex <id_list>
- Direction Curve <id>
- Segment|Fraction|Distance <val> [From Curve <id>]
- Through Vertex <id_list>
- Parametric <on|OFF>
- Tolerance <val>
- Preview [Create]
- Settings (Tolerance, Parametric, Triangle)

The volume shown in Figure 4 was quickly prepared for sweeping by splitting the fillets and specifying sweep sources as shown (with the sweep target underneath the volume). The surface splits are shown in blue.

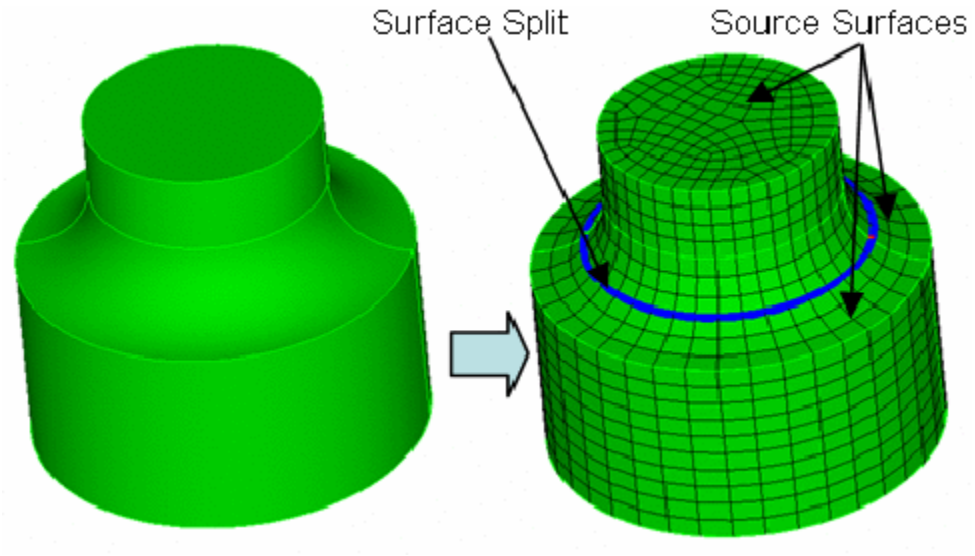


Figure 4 - Splitting Fillets to Facilitate Sweeping

Each surface is always split with a *single* curve along the length of the surface (or multiple single curves if the Segment option is used). The splitting curve will either be a spline, arc or straight line.

Logical Rectangle

The Split Surface command analyzes the selected surface or surface chain to find a *logical rectangle*, containing four logical sides and four logical corners; each side can be composed of zero, one or multiple curves. If a single surface is selected (with no options), the logical corners will be those closest to 90° and oriented such that the surface will be ***split parallel to the longest aspect ratio*** of the surface. If a chain of surfaces is selected, the logical corners will include the two corners closest to 90° on the starting surface of the chain and the two corners closest to 90° on the ending surface of the chain (***the split will always occur along the chain***).

In Figure 5, the logical corners selected by the algorithm are Vertices 1-2-5-6. Between these corner vertices the logical sides are defined; these sides are described in Table 1. The default split occurs from the center of Side 1 to the center of Side 3 (parallel to the longest aspect ratio of the surface), and is shown in blue.

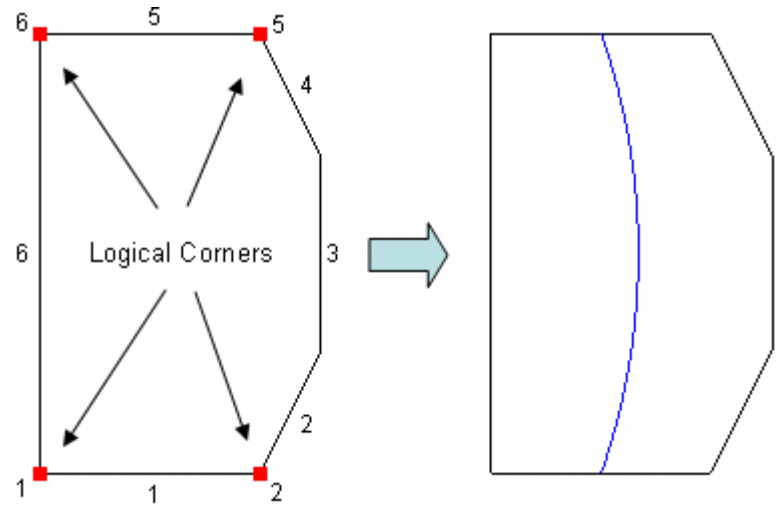


Figure 5 - Split Surface Logical Properties

Table 1. Listing of Logical Sides for Figure 5

Logical Side	Corner Vertices	Curve Groups
1	1-2	1
2	2-5	2,3,4
3	5-6	5
4	6-1	6

Figure 6 shows a surface along with 2 possibilities for its logical rectangle and the resultant splits.

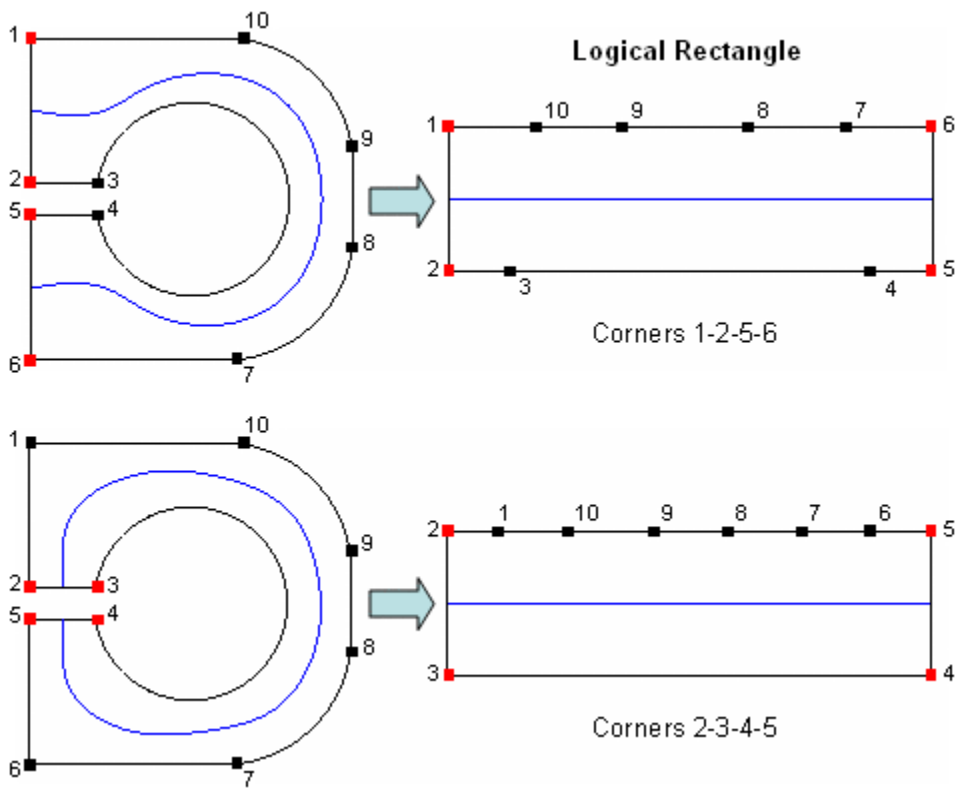
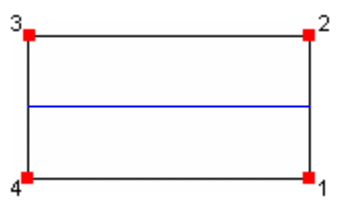
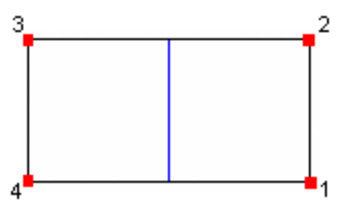
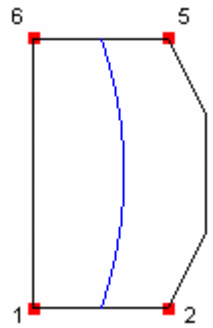
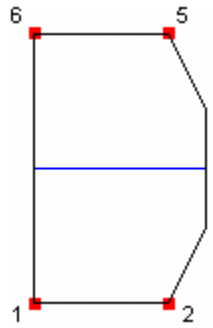
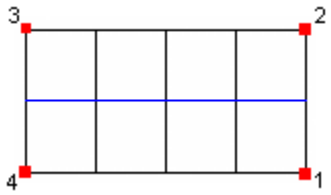
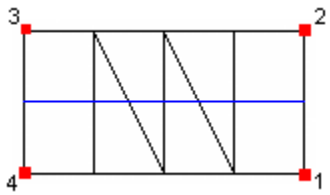
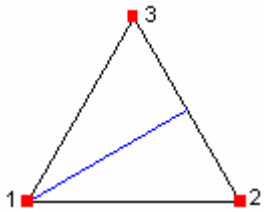


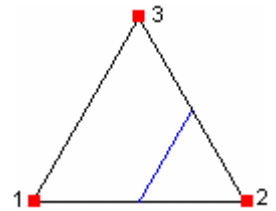
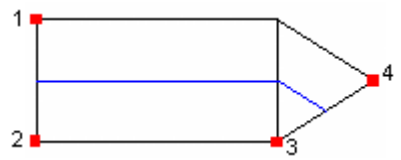
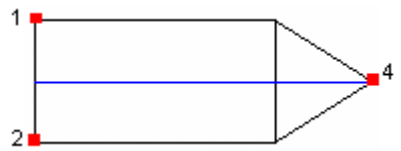
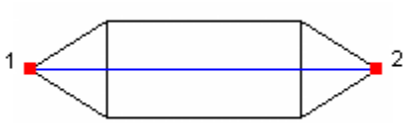
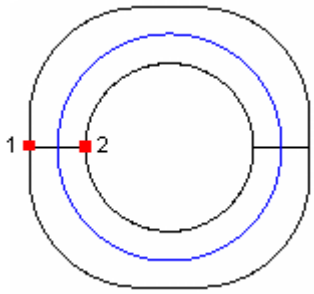
Figure 6 - Different Possible Logical Rectangles for Same Surface

Table 2 shows various surfaces and the resultant split based on the automatically detected or selected logical rectangle. Note that surfaces are always traversed in a counterclockwise direction.

Table 2 - Sample Surfaces and Logical Rectangles

Surface(s) (Resultant Split in Blue)	Ordered Corners (to form the *Logical Rectangle*)
	1-2-3-4 (using aspect ratio)
	4-1-2-3 (user selected)

	1-2-5-6
	2-5-6-1
	1-2-3-4 (split is always along the chain)
	1-2-3-4 (notice triangular surfaces along the chain)
	1-1-2-3 (note side 1 of the logical rectangle is collapsed; side 3 is from vertex 2 to 3)

	1-2-2-3 (note side 2 of the logical rectangle is collapsed)
	1-2-3-4
	1-2-4-4
	1-1-2-2
	1-1-2-2 (selected automatically)

Split Orientation

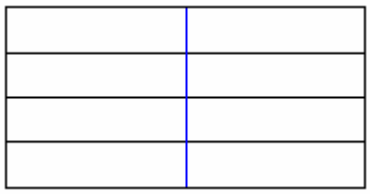
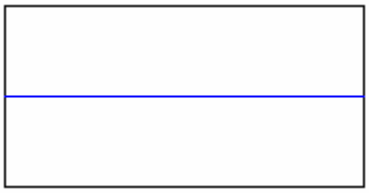
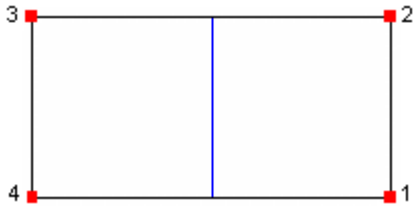
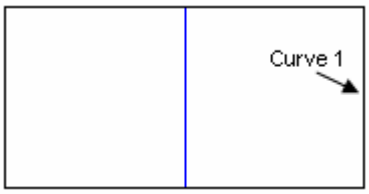
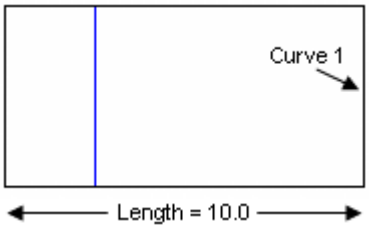
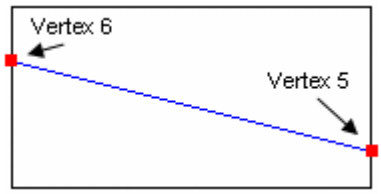
If a chain of surfaces are split, the surfaces will always be split along the chain. The command will not allow disconnected surfaces.

For a single surface, the split direction logic is a bit more complicated. If no options are specified, the surface aspect ratio determines the split direction - the surface will be split parallel to the longest aspect ratio side through the midpoint of each curve. This behavior can be overridden by the order the Corner vertices are selected (the split always starts on the side between the first two corners selected), the Direction option, the From Curve option, or the Through Vertex list.

Table 3 shows examples of the various split orientation methods. These options are explained in more detail in the sections below.

Table 3 - Split Orientation Methods

Surface Example	Split Orientation Method

	Multiple surfaces are *always* split along the chain
	Parallel to longest surface aspect ratio (default)
	Corner Vertex 4 1 2 3 (split always starts on side 1 of the logical rectangle)
	Direction Curve 1
	From Curve 1 Fraction .75 or From Curve 1 Distance 7.5
	Through Vertex 5 6

Corner Specification

The **Corner** option allows you to specify corners that form *logical rectangle* the algorithm uses to orient the split on the surface. When analyzing a surface to be split, the software automatically selects the corners that are closest to 90°. The Preview option displays the automatically selected corners in red. Sometimes incorrect corners are chosen, so you must specify the desired corners yourself. The split always starts on the side between the first two corners selected and finishes on the side between the last two corners selected. Figure 7 shows a situation where the user had to select corners to get the desired split.

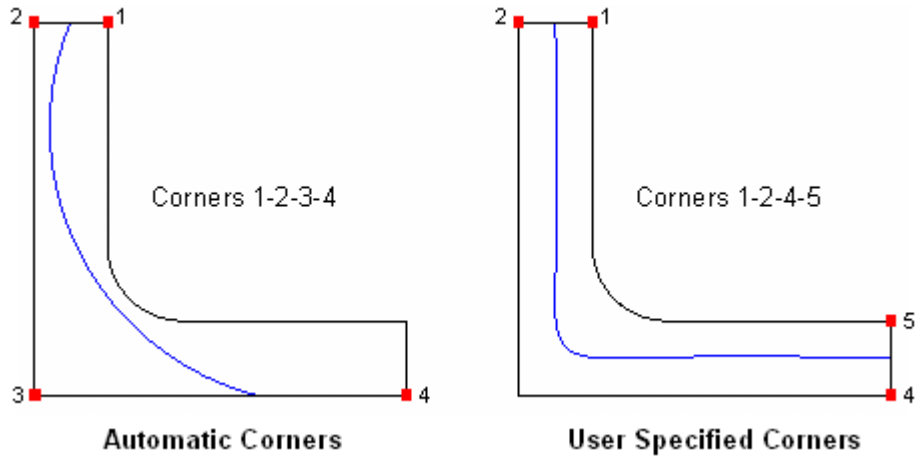
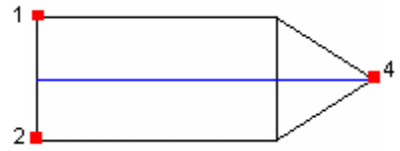
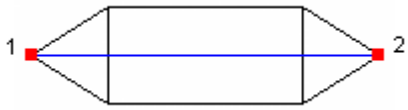


Figure 7 - Selecting the Desired Corners

The split can be directed to the tip of a triangular shaped surface by selecting that corner vertex twice (at the start or end of the corner list) when specifying corners, creating a zero-length side on the logical rectangle. A shortcut exists whereas if you specify only 3 corner vertices, the zero-length side will be directed to the first corner selected. If you specify only 2 corner vertices, a zero-length side will be directed to both the first and second corner you select. Table 4 shows these examples. Note the software will automatically detect triangle corners based on angle criteria - the corner selection methods for zero-length sides explained in this section need only be applied if the angles are outside of the thresholds specified in the Set Split Surface Auto Detect Triangle settings.

Table 4 - Selecting Corners to Split to Triangle Tips

Surface	Corner Specification
	1-2-4-4- or 4-4-1-2 or 4-1-2 (shortcut method)
	1-1-2-2 or 2-2-1-1 or 1-2 or 2-1 (shortcut method)

Direction

The **Direction** option allows you to conveniently override the default split direction on a single surface. Simply specify a curve from the logical rectangle that is parallel to the desired split direction. If Corners are also specified, the Direction option will override the split orientation that would result from the specified corner order. The Direction option is not valid on a chain of surfaces. Figure 8 shows an example.

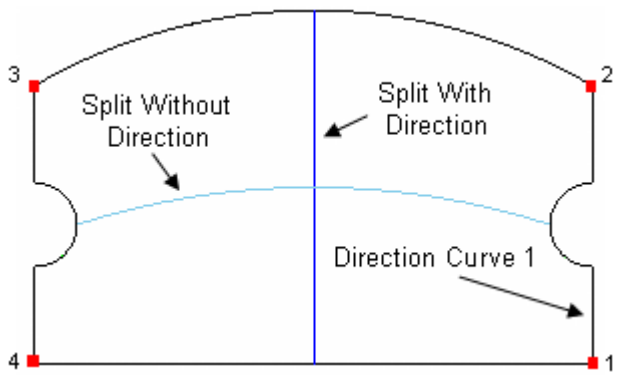
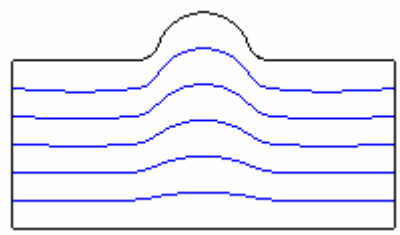
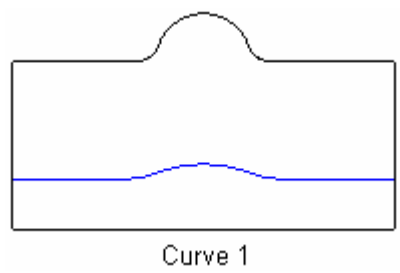
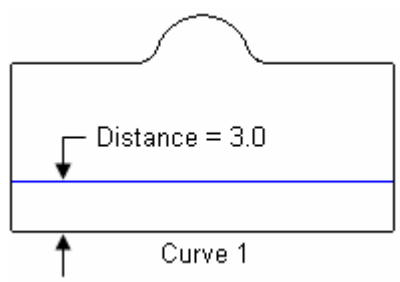


Figure 8 - Direction Specification Overrides Corner Order

Segment|Fraction|Distance

The **Segment** option allows you to split a surface into 2 or more segments that are equally spaced across the surface. The **Fraction** option allows you to override the default 0.5 fractional split location. The **Distance** option allows you to specify the split location as an absolute distance rather than a fraction. By specifying a **From Curve**, you can indicate which side of the logical rectangle to base the segment, fraction or distance from (versus a random result). Table 5 gives examples of these options.

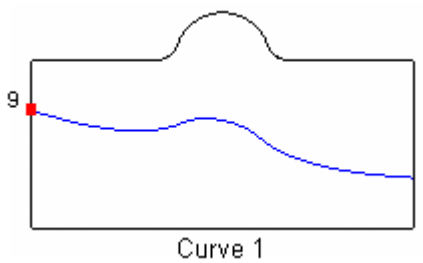
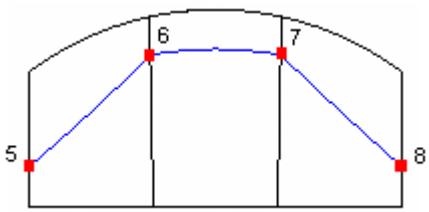
Table 5 - Segment, Fraction, Distance Examples

Surface	Command Options
	Segment 6 From Curve 1
	Fraction .3 From Curve 1
	Distance 3 From Curve 1

Through Vertex

The **Through Vertex** option forces the split through vertices on surface boundaries perpendicular to the split direction. Use this option if the desired fraction is not constant from one end of the surface to another or if a split would otherwise pass very close to an existing curve end resulting in a short curve. Through vertices can be used in conjunction with the Fraction option - the split will linearly adjust to pass exactly through the specified vertices. It is not valid with the Segment option. The maximum number of Through Vertices that can be specified is equal to the number of surfaces being split plus one. The selected vertices can be free, but must lie on the perpendicular curves. Table 6 gives several examples.

Table 6 - Through Vertex Examples

Surface(s)	Command Options
	Fraction .3 From Curve 1 Through Vertex 9
	Through Vertex 5 6 7 8

Parametric

By default, split locations are calculated in 3D space and projected to the surface. As an alternative, split locations can be calculated directly in the surface parametric space. In rare instances, this can result in a smoother or more desirable split. The command option **Parametric {on|Off}** can be used to split the given surfaces in parametric space. Alternatively, the default can be overridden with the Set Split Surface Parametric {on|OFF} command.

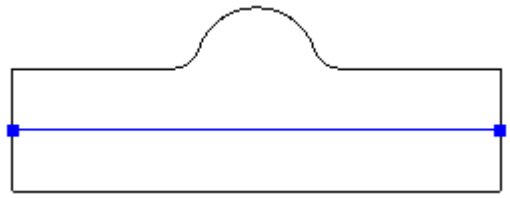
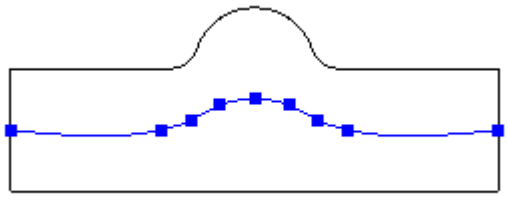
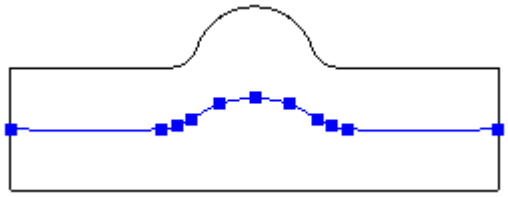
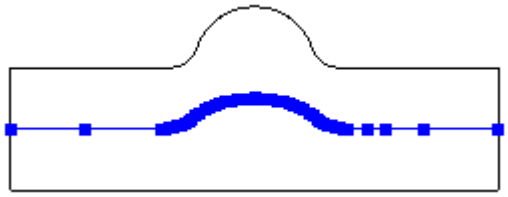
Tolerance

A single absolute tolerance value is used to determine the accuracy of the split curves. A smaller tolerance will force more points to be interpolated. The tolerance is also used when detecting an analytical curve (e.g., an arc or straight line) versus a spline. A looser tolerance will result in more analytical curves. The default tolerance is 1.0. The command option **Tolerance <val>** can be used to split the given surfaces using the given tolerance. Alternatively, the default tolerance can be overridden with the Set Split Surface Tolerance <val> command.

It is recommended to use the largest tolerance possible to increase the number of analytical curves and reduce the number of points on splines, resulting in better performance and smaller file sizes. The Preview option displays the interpolated curve points. Table 7 shows the effect of the tolerance for a simple example.

Table 7 - Effect of Tolerance on Split Curve

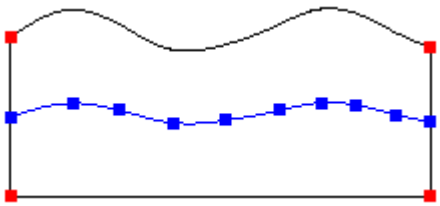
Surface	Tolerance

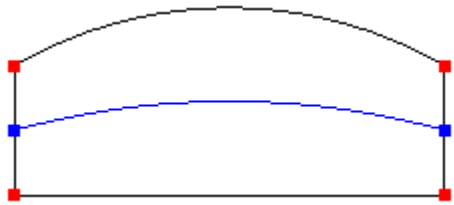
	2.0
	1.0
	0.5
	0.01

Preview

The **Preview** keyword will show a graphics preview (in blue) of the splitting curve (or curves) and the corner vertices (in red) selected for the logical rectangle. The curve preview includes the interpolated point locations that define spline curves. Note that if no points are shown on the interior of the curve, it means that the curve is an analytical curve (line or arc). If the **Create** keyword is also specified, a free curve (or curves) will be created - these are the internal curves that are used to imprint the surfaces. Table 8 shows some examples.

Table 8 - Graphics Preview

Surface	Curve Type
	Spline

	Arc (no preview points shown on interior of curve)

Settings

This section describes the settings that are available for the automatic split surface command. To see the current values, you can enter the command **Set Split Surface**, optionally followed by the setting of interest (without specifying a value).

Set Split Surface Tolerance <val>

This sets the default tolerance for the accuracy of the split curves. See the Tolerance section for more information.

Set Split Surface Parametric {on|OFF}

This sets the default for whether surfaces are split in 3D (default) or in parametric space. See the Parametric section for more information.

Set Split Surface Auto Detect Triangle {ON|off}

Set Split Surface Point Angle Threshold <val>

Set Split Surface Side Angle Threshold <val>

The split surface command automatically detects triangular shaped surfaces as explained in the section on Corners. This behavior can be turned off with the setting above. Two thresholds are used when detecting triangles - the **Point Angle** threshold and the **Side Angle** threshold, specified in degrees. Corners with an angle below the Point Angle threshold are considered for the tip of a triangle (or the collapsed side of the logical rectangle). Corners within the Side Angle threshold of 180° are considered for removal from the logical rectangle. In order for a triangle to actually be detected, corners for both the point and side criteria must be met. The default Point Angle threshold is 45°, and the default Side Angle threshold is 27°. Figure 9 provides an illustration.

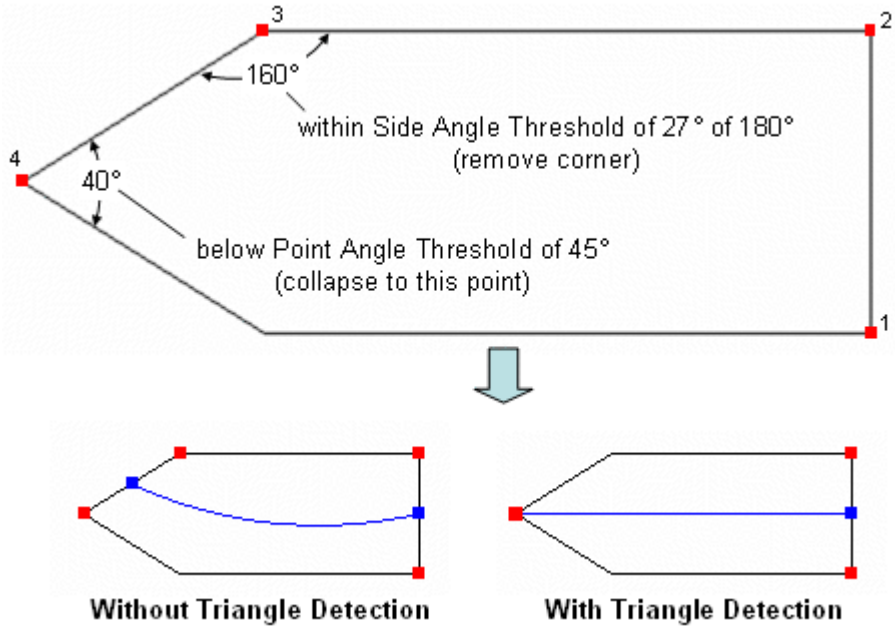


Figure 9: Triangle Detection Settings

Split Periodic Surfaces

Solids which contain periodic surfaces include cylinders, torii and spheres. Splitting periodic surfaces can in some cases simplify meshing, and will result in curves and surfaces being added to the volume. The command used to split periodic surfaces is:

Split Periodic Body {id_range|all}

This command splits all periodic surfaces in a body or bodies.

Separating Multi-Volume Bodies

The separate and split commands are used to separate a body with multiple volumes into a multiple bodies with single volumes. The commands are:

Separate {Body|Volume} {id_range|all}

and

Split {Body|Volume} {id_range|all}

Only very rarely will either of these commands be needed. They are provided for the occasional instance that a multi-volume body is found. These commands are interchangeable.

Section Command

This command will cut a body or group of bodies with a plane, keeping geometry on one side of the plane and discarding the rest. The syntax for this command is:

Section {body|group} <id_range> [with] {xplane|yplane|zplane} [offset <value>] [NORMAL|reverse] [keep]

Section {body|group} <id_range> with surface <id> [NORMAL|reverse] [keep]

In the first form, the specified coordinate plane is used to cut the specified bodies. The offset option is used to specify an offset from the coordinate plane. In the second form, an existing (planar) surface is used to section the model. In either case, the reverse keyword results in discarding the positive side of the specified plane or surface instead of the other side. The keep option results in keeping both sides; the section command used with this option is equivalent to webcutting with a plane.

Geometry Imprinting and Merging

- Imprinting Geometry
- Merging Geometry
- Examining Merged Entities
- Merge Tolerance
- Unmerging
- Using Geometry Merging to Verify Geometry

Geometry is created and imported in a manifold state. The process of converting manifold to non-manifold geometry is referred to as "geometry merging", since it involves merging multiple geometric entities into single ones. When importing mesh-based geometry, the merging step can be automatic. Imprinting is a necessary step in the merging process, which ensures that entities to be merged have identical topology.

Imprinting Geometry

To produce a non-manifold geometry model from a manifold geometry, coincident surfaces must be merged together (See Geometry Merging); this merge can only take place if the surfaces to be merged have like topology and geometry. While various parts of an assembly will typically have surfaces, which coincide geometrically, an imprint is necessary to make the surfaces have like topology. To preview which surfaces can or should be imprinted, or to force imprints that the regular imprint command misses, the Find Overlap command can be used.

The commands used to imprint bodies together are:

Imprint <body1_id> with <body2_id>

A body can also be imprinted with curves, or vertices and surfaces can be imprinted with curves. It is useful to imprint bodies or surfaces with curves to eliminate mesh skew, generate more favorable surfaces for meshing, or create hard lines for paving. Imprinting with a vertex can be useful to split curves for better control of the mesh or create hard points for paving.

Imprint Body <body_id_range> [with] Curve <curve_id_range> [keep]

Imprint Body <body_id_range> [with] Vertex <vertex_id_range> [keep]

Imprint Surface <surface_id_range> [with] Curve <curve_id_range> [keep]

An **Imprint All** will imprint all bodies in the model pairwise; bounding boxes are used to filter out imprint calls for bodies which clearly don't intersect.

Imprint [Body] All

Another form of the imprint command,

Imprint Mesh {Body | Volume} <id_list>

uses coincident mesh entities and virtual geometry to create imprints. See the Partitioned Geometry section for more information on this command.

Merging Geometry

The steps of the geometry merging algorithm used in CUBIT are outlined below:

1. Check lower order geometry, merge if possible
2. Check topology of current entities
3. Check geometry of current entities
4. If both entities are meshed, check topology of meshes.
5. If geometric topology, geometry, and mesh topology are alike, merge.

Thus, in order for two entities to merge, the entities must correspond geometrically and topologically, and if both are meshed must have topologically equivalent meshes. The geometric correspondence usually comes from constructing the model that way. The topological correspondence can come from that process as well, but also can be accomplished in CUBIT using Imprinting.

If both entities are meshed, they can only be merged if the meshes are topologically identical. This means that the entities must have the same number of each kind of mesh entity, and those mesh entities must be connected in the same way. The mesh on each entity need not have nodes in identical positions. If the node positions are not identical, the position of the nodes on the entity with the lowest ID will be used in the resulting merged mesh.

There are several options for merging geometry in CUBIT.

Merge geometry automatically

Merge All [Group|Body|Surface|Curve|Vertex] [group_results]

All topological entities in the model or in the specified bodies are examined for geometric and topological correspondence, and are merged if they pass the test.

If a specific entity type is specified with the Merge all, only complete entities of that type are merged. For example, if Merge all surface is entered, only vertices which are part of corresponding surfaces being merged; vertices which correspond but which are not part of corresponding surfaces will not be merged. This command can be used to speed up the merging process for large models, but should be used with caution as it can hide problems with the geometry.

Test for merging in a specified group of geometry

Merge {Group|Body|Surface|Curve|Vertex} <id_range>[With {Group|Body|Surface|Curve|Vertex} <id_range>][group_results]

All topological entities in the specified entity list, as well as lower order topology belonging to those entities, are examined for merging. This command can be used to prevent merging of entities which correspond and would otherwise be merged, e.g. slide surfaces.

Force merge specified geometry entities

Merge Vertex <id> with Vertex <id> Force

Merge Curve <id> with Curve <id> Force

Merge Surface <id> with Surface <id> Force

This command results in the specified entities being merged, whether they pass the geometric correspondence test or not. This command should only be used with caution and when merging otherwise fails; instances where this is required should be reported to the CUBIT development team.

Preventing geometry from merging

Body <id_range> Merge [On | Off]

Volume <id_range> Merge [On | Off]

Surface <id_range> Merge [On | Off]

Curve <id_range> Merge [On | Off]

Vertex <id_range> Merge [On | Off]

These commands provide a method for preventing entities from merging. If merging is set to off for an entity, merging commands (e.g. "merge all") will not merge that entity with any other.

Examining Merged Entities

There are several mechanisms for examining which entities have been merged. The most useful mechanism is assigning all merged or unmerged entities of a specified type to a group, and examining that group graphically. This process can be used to examine the outer shell of an assembly of volumes, for example to verify if all interior surfaces have been merged. To put all the merged or unmerged entities of a given type into a specified group, use the command:

Group {<`name'>|<id>} [Surface | Curve | Vertex] [Merged | Unmerged]

If the entity type is unspecified, surfaces will be assumed.

Entities can also be labelled in the graphics according to the state of their merge flag. See the Preventing geometry from merging section for information on controlling the merge flag. To turn merge labeling on for a specified entity type, use the command

Label {Vertex | Curve | Surface} Merge

Merge Tolerance

Geometric correspondence between entities is judged according to a specified absolute numerical tolerance. The particular kind of spatial check depends on the type of entity. Vertices are compared by comparing their spatial position; curves are tested geometrically by testing points 1/3 and 2/3 down the curve in terms of parameter value; surfaces are tested at several pre-determined points on the surface. In all cases, spatial checks are done comparing a given position on one entity with the closest point on the other entity. This allows merging of entities which correspond spatially but which have different parameterizations.

The default absolute merge tolerance used in CUBIT is 5.0e-4. This means that points which are at least this close will pass the geometric correspondence test used for merging. The user may change this value using the following command:

Merge Tolerance <val>

If the user does not enter a value, the current merge tolerance value will be printed to the screen. There is no upper bound to the merge tolerance, although in experience there are few cases where the merge tolerance has needed to be adjusted upward. The lower bound on the tolerance, which is tied to the accuracy of the solid modeling engine in CUBIT, is 1e-6.

Unmerging

The unmerge command is used to reverse the merging operation. This is often in cases where further geometry decomposition must be done.

unmerge all

unmerge <entity_list>

Un-merging an entity means that the specified geometric entity and all lower-order (or child) entities will no longer share non-manifold topology with any other entities. For example, if a body is unmerged, that body will no longer share any surfaces, curves, or vertices with any other body.

[set] unmerge duplicate_mesh {on|off}

If any meshed geometry is unmerged, the mesh is kept as necessary to keep the mesh of higher-order entities valid. For example, if a surface shared by two volumes is to be unmerged and only one of the volumes is meshed, the surface mesh will remain with whichever surface is part of the meshed volume.

When unmerging meshed entities, the default behavior of the code is that the placement if the mesh is determined by the following rules:

- If neither entity has meshed parent entities, the mesh is kept on one of the two entities.
- If one entity has a meshed parent entity, the mesh is kept on that entity.
- If both entities have meshed parents, the mesh is kept on one and copied on the other.

If **unmerge duplicate_mesh** is turned on, the rules described above are overwritten and whenever a meshed entity is unmerged the mesh is always copied such that both entities remain meshed.

To get back to the default behavior, turn **unmerge duplicate_mesh** off.

Using Geometry Merging to Verify Geometry

Geometry merging is often used to verify the correctness of an assembly of volumes. For example, groups of unmerged surfaces can be used to verify the outer shell of the assembly (see Examining Merged Entities.) There is other information that comes from the **Merge all** command that is useful for verifying geometry.

In typical geometric models, vertices and curves which get merged will usually be part of surfaces containing them which get merged. So, if a **Merge all** command is used and the command reports that vertices and curves have been merged, this is usually an indication of a problem with geometry. In particular, it is often a sign that there are overlapping bodies in the model. The second most common problem indicated by merging curves and vertices is that the merge tolerance is set too high for a given model. In any event, merged vertices and curves should be examined closely.

Virtual Geometry

- Composite Geometry
- Partitioned Geometry
- Collapse Angle
- Deleting Virtual Geometry

The Virtual Geometry module in CUBIT provides a way to modify the topology of the model without affecting the underlying ACIS geometry representation and without making changes to the actual solid model. Virtual Geometry includes the capability to composite or partition geometry as well as creates new virtual geometric entities. Virtual Geometry operations are most often used as a tool for adjusting the geometry to allow mapping, sub-mapping or sweeping mesh generation schemes to be applied.

The advantage to using Virtual Geometry is that all operations are reversible. With standard geometry modification commands, changes are made to the underlying geometry representation and cannot be changed once effected. With virtual geometry, the original solid model topology can be easily restored. This is useful when geometry modifications are made in order to apply a particular meshing scheme. Virtual geometry can be applied and later removed once the part has been meshed.

It should be noted that geometry modification operations such as webcutting, imprinting or booleans on the solid model cannot be done for bodies containing virtual geometry. Solid modeling operations must be done prior to creating virtual geometry.

Composite Geometry

- Composite Curves
- Composite Surfaces

The virtual geometry module has the capability to combine a set of connected curves into a single composite curve, or a set of connected surfaces into a single surface. The general purpose is to suppress or remove the child geometry common to those entities being composited. For example, compositing a set of curves suppresses the vertices common to those curves, thus removing the constraint that a node must be placed at that vertex location.

The basic form of the command to create composites is:

```
composite create {surface|curve} <id_list>
```

This command will composite as many surfaces (or curves) as possible, in many cases creating multiple composites.

The entities combined to create the composite must either all be unmeshed or all be meshed. A meshed composite surface can not be removed unless the mesh is first deleted.

Care should be taken when compositing over large C^1 discontinuities as it may cause problems for the meshing algorithms and may result in poor quality elements. C^1 discontinuities are corners or abrupt changes in the surface normal.

The command to remove a composite is:

composite delete {surface|curve} <id>

Composite Curves

The full command for the creation of composite curves is:

Composite Create Curve <id_range> [keep vertex <id_list>] [angle <degrees>]

The additional arguments provide two methods to prevent vertices from being removed from the model or *composited* over. The first method, **keep vertex** explicitly specifies vertices which are not to be removed. This option can also be used to control which vertex is kept when compositing a set of curves results in a closed curve.

The **angle** option specifies vertices to keep by the angle between the tangents of the curves at that vertex. A value less than zero will result in no composite curves being created. A value of 180 or greater will result in all possible composites being created. The default behavior is an empty list of vertices to keep, and an angle of 180 degrees.

Composite Surfaces

The general command for composite surface creation is:

Composite Create Surface <id_range> [angle <degrees>] [nocurves] [keep [angle <degrees>] [vertex <id_list>]]

The **angle** argument prevents curves from being removed from the model or *composited* over. Composites will not be generated where the angle between surface normals adjacent to the curve is greater than the specified angle.

When a composite surface is created, the default behavior is to also to composite curves on the boundary of the new composite surface.

Curves are automatically composited if the angle between tangents at the common vertex is less than 15 degrees. The **nocurve**s option can be used to prevent any composite curves from being created.

The **keep** keyword can be used to change the default choice of which curves to composite. The arguments following the **keep** keyword behave the same as for explicit composite curve creation. The **nocurves** and **keep** arguments are mutually exclusive.

Controlling the Surface Evaluation Method for Composite Surfaces

It typically takes longer to mesh a single composite surface than to mesh the surfaces used in the creation of the composite. To improve speed, composite surfaces use an approximation method to evaluate the closest point to a trimmed surface. However, this evaluation method may give poor results for composites of highly convoluted surfaces.

The virtual geometry module provides a way to change the way surfaces are evaluated using the following command:

Composite closest_pt surface <id> {gme|emulate}

The default behavior is to use the **emulate** method, as it is typically considerably faster. Specifying the **gme** option will force the specified composite surface to use the exact calculation of the closest point to a trimmed surface, as provided by the solid modeler. The **gme** option, however, can be considerably slower.

Composite Determination

The **composite create surface** command is non-deterministic in some circumstances. When three or more adjacent surfaces are to be composited, all the surfaces may not be able to be composited into a single surface as illustrated in Figure 1. In this case different subsets of the surfaces may be composited and the command will choose arbitrary subsets to composite. As an example, there are three surfaces A, B, and C, all adjacent to each other. The common curve between A and B is AB, the common curve between B and C is BC, and the common curve between A and C is CA. If the curve BC cannot be removed, either due to the angle specified in the composite command, or because there is a fourth surface, D, also using that curve, the command will arbitrarily choose to either composite A and B or A and C.

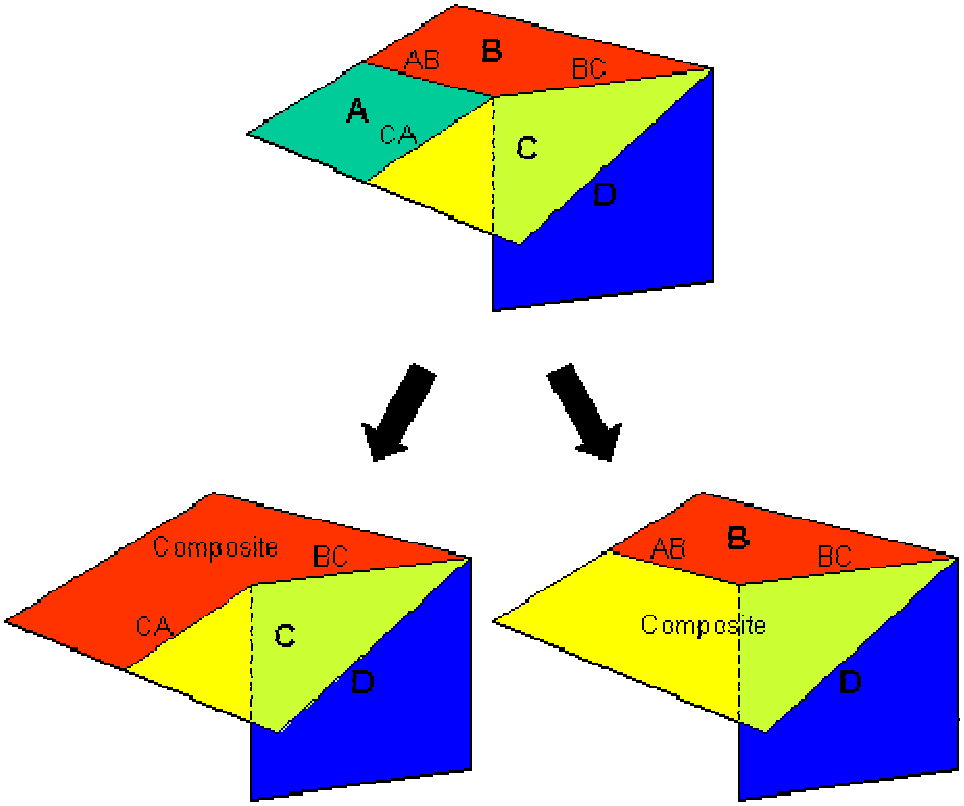


Figure 1. In some cases, the program will make a determination of which surfaces to composite.

Partitioned Geometry

Partitioning provides a method to introduce additional topology into the model, to better constrain meshing algorithms. This is accomplished by splitting, or partitioning, existing curves or surfaces.

- Partitioned Curves
- Partitioned Surfaces
- Partitioned Volumes
- Using Mesh Intersections to Partition Surfaces
- Removing Partitions

Partitioned Curves

There are four methods for specifying locations at which to partition curves:

Partition Create Curve <curve_id> {Fraction <fraction_list> | Position <xpos> <ypos> <zpos> | [with] <vertex_list> | <node_list> }

The first two forms of the command create additional vertices and use those vertices to split a curve. The third form of the command uses existing vertices to split the curve. The fourth form of the command uses existing nodes to split the curve.

Using the **fraction** option, vertices are created at the specified fractions along the curve (in the range [0,1].) Subsequently, the curve is split at each vertex, resulting in n+1 new curves, where n is the number of fraction values specified.

Using the **position** option, vertices are created at the closest location along the curve to each of the specified position. Subsequently, the curve is split at each vertex, resulting in n+1 new curves, where n is the number of positions specified.

If the **node** option is used, meshed curves may be partitioned. The specified nodes must lie on the curve to be partitioned. The curve is split at each node specified, and any other mesh entities are divided appropriately amongst the curve partitions.

Partitioned Surfaces

There are several forms of the command to partition a surface. A surface may be partitioned using hard points, curves, polylines, mesh edges, mesh faces or mesh triangles.

- Partitioning with Vertices or Nodes
- Partitioning with Curves
- Partitioning with Mesh Edges
- Partitioning with Mesh Faces or Triangles

Partitioning with Vertices and Nodes

Partitioning with Hard Points

There are two methods of partitioning a surface using vertices and nodes. The first method is to create a set of hard points using nodes, vertices, or coordinates that constrain the mesh to particular points on the surface. The syntax is:

Partition create surface <id> vertex <id_list> [individual]

Partition create surface <id> node <id_list> [individual]

Partitioning with Polylines

The second method is to define a polyline using a set of vertices or coordinates. This method splits the surface using a polyline defined by the a list of positions specified as either coordinate triples, or existing vertices. The polyline is projected to the surface to define the curve for splitting the surface. If only one position is specified a zero-length curve with a single vertex will be created The syntax is identical to above WITHOUT the individual option.

Partition create surface <id> vertex <id_list>

Partition create surface <id> position <x> <y> <z> [[Position] <x> <y> <z> ...]

In the following simple example, the surface is partitioned using both methods. On the left half of the object, the surface is partitioned using the individual option (vertices 11 12 15 13). On the right half, a polyline is used (vertices 9 10 16 14). All of the free vertices can then be deleted, leaving the virtual curves shown in the second picture. Vertices 19 20 21 and 22 are all zero-length curves. The small 'v' in parentheses is to indicate that it is virtual geometry. The resulting mesh is shown in the third picture. Notice that the polyline constrains the entire curve to the mesh, while the hardpoints constrain only that individual point.

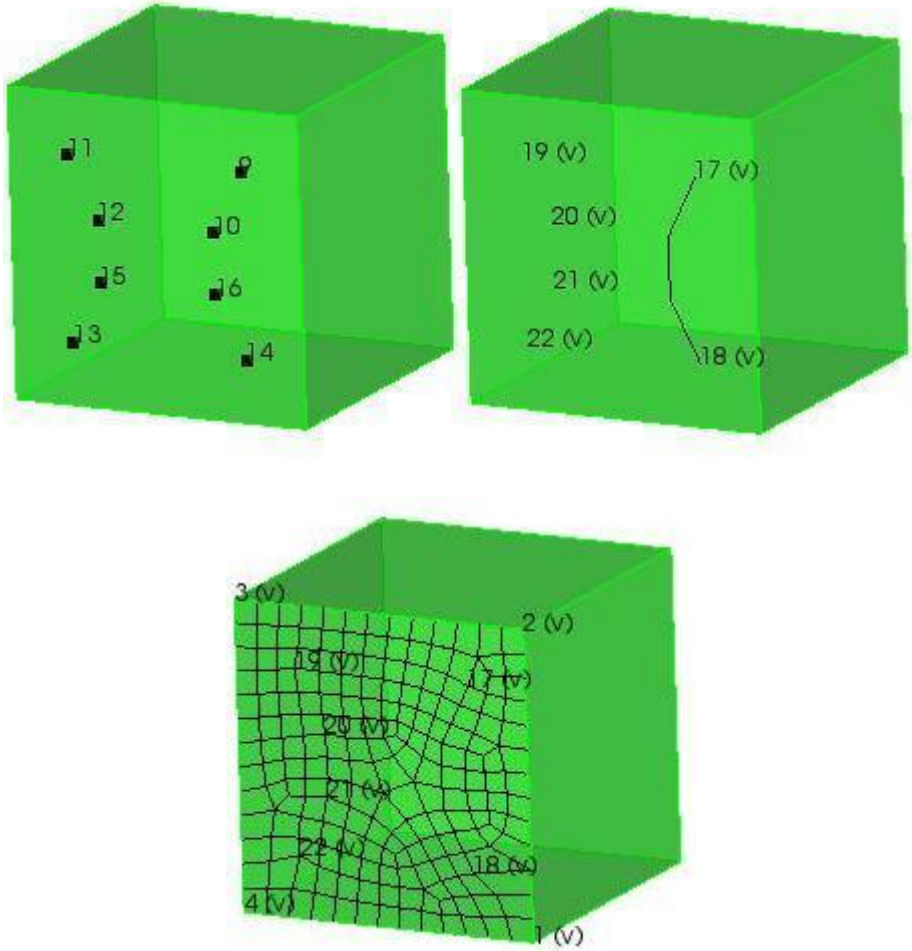


Figure 1. Partitioning a Surface Using Vertices

Partitioning with Curves

This form of the command splits the existing surface into several surfaces by creating curves that approximate the projection of the specified existing curves onto the surface. The syntax is:

Partition create surface <id> curve <id_list>

Partitioning with Mesh Edges

Meshed surfaces may be partitioned with mesh edges. The specified mesh edges must be owned by the surface to be partitioned. The shape of the curve(s) used to split the surface is specified by a set of mesh edges.

If the split location is specified by a series of mesh edges, and the specified mesh edges form a closed loop, the node option may be used to control which node the vertex is created at.

Partition create surface <id> edge <id_list> [Node <node_id>]

Partitioning with Faces or Triangles

Surfaces may also be partitioned by specifying a list of triangles or faces (quads). The boundary of the list will automatically be detected and new curves and vertices created at the appropriate locations. Curves are created from the mesh edges and used to split the surface. The surface mesh is split and assigned to the appropriate surface partitions.

Partition create surface <id> face|tri <id_list>

Partitioned Volumes

To partition a volume by giving a center and radius:

Partition create volume <id> center [Location] {options} radius <val>

This command splits the existing volume into two volumes. All volume elements that lie within the specified radius of the specified center location are identified, and the exterior faces of these elements are used to create a surface and partition the volume. The center can be specified with any of the location options.

Figure 1 shows an example of a partitioned volume. A cube that has been map meshed is partitioned using a center at one of its vertices. The result is two distinct volumes with a surface separating the two. The interface surface is composed of the faces of the interior hex elements.

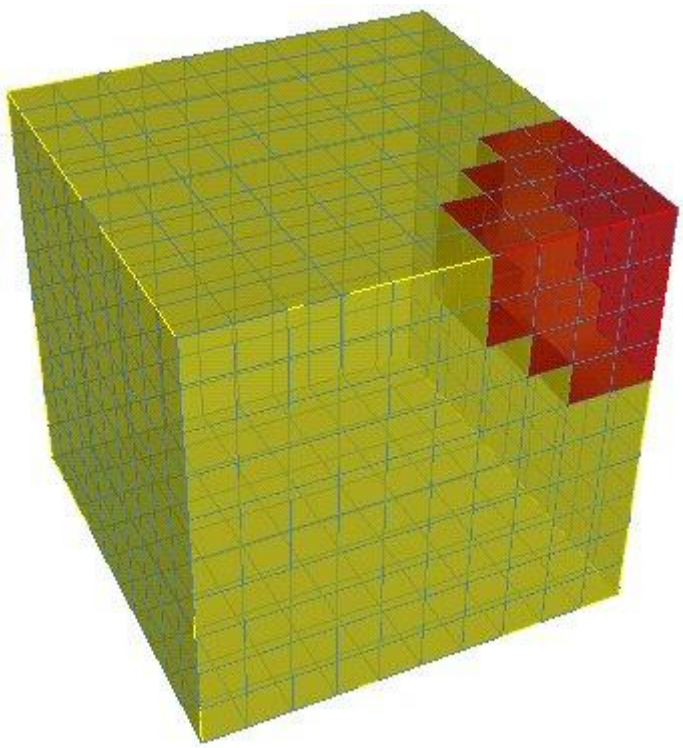


Figure 1. A partitioned volume

This command may be useful for separating small regions of a meshed volume so that remeshing or mesh improvement may be performed locally.

Using Mesh Intersections to Partition Surfaces

To assist in various mesh editing tasks such as joining, a *mesh-based imprinting* capability is provided. The command

Imprint Mesh {Body | Volume} <id_list>

determines imprint locations using the mesh on the surfaces of the specified bodies or volumes. Regions of coincidence between the surfaces is determined by searching for coincident nodes in the mesh of the surfaces. Virtual geometry is then used to partition the surfaces and curves at the boundary of these regions of coincident mesh.

The **imprint mesh** functionality differs from a normal geometric imprint in the following ways:

- The location of the imprint is determined from coincidence of mesh nodes.
- The mesh remains intact through the imprint operation.
- Virtual geometry is used to create the imprint.
- The imprinting can be done on all types of geometry (including mesh-based geometry, merged geometry, and virtual geometry.)

The following is a trivial example of this capability. The following commands create two meshed blocks:

```
brick width 10
brick width 6
body 2 move x 8
volume 1 2 size 1
mesh volume 1 2
```

Figure 1 shows the results of these commands.

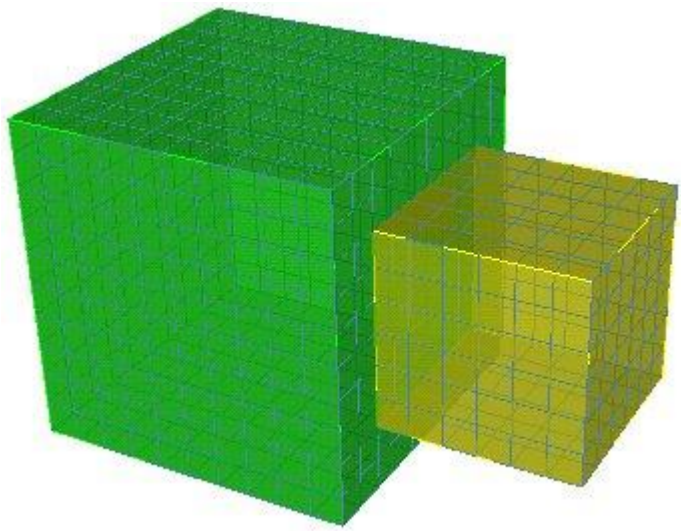


Figure 1. Two adjacent meshed volumes. The coincident meshes will form the basis of the imprint operation.

The mesh of the blocks can be joined by first doing a mesh-based imprint and then merging:

```
imprint mesh body 1 2
merge body 1 2
```

Figure 2. shows the results of the imprint operation. A meshed surface is created at the interface between the two meshed volumes. The nodes on the new surface are shared by the neighboring hexahedra of both volumes.

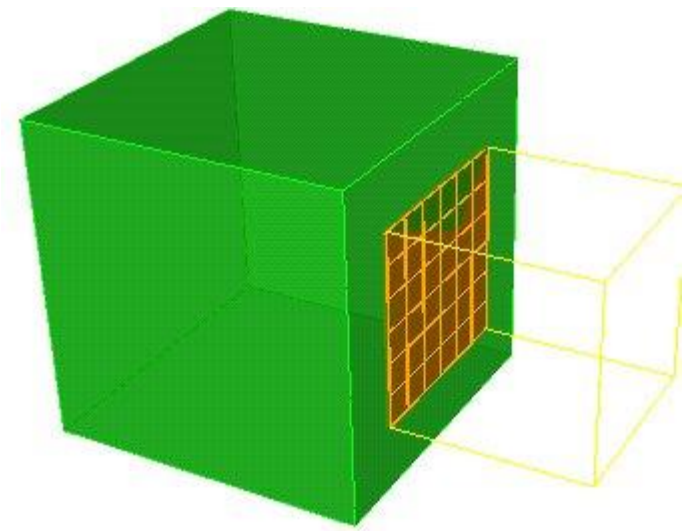


Figure 2. The imprinted surface. Adjacent volume meshes joined at the interface surface.

Use of this functionality is discussed further in the Mesh Editing section of the Appendix

Removing Partitions

There are two commands used to remove partitions:

Partition merge {curve|surface|volume} <id_list>

The command combines existing partitions where possible. This command is similar to the composite create command. The difference is that this command is special-cased for partitions, and will result in more efficient geometric evaluations. If all the partitions of a real solid model entity are merged, such that there is only one partition remaining, the virtual geometry will be removed, and the original solid model geometry will be restored to the model.

The CUBIT delete command can also be used for removing partitions. See Deleting Virtual Geometry for a description of its use.

Collapse Angle

The collapse command allows the user to collapse small angles using virtual geometry. The command syntax is:

Collapse angle at vertex <id> curve <id1> [arc_length1 <length>] curve <id2> [arc_length2 <length> | same_size | perpendicular | tangent] [composite_vertex <angle>] [preview]

The collapse angle command is used to eliminate small angles at vertices, where curves meet at a tangential point. The command will split each curve at a specified distance (δ1 and δ2) as shown in Figure 1, and create two new vertices along those curves. The remaining small angle will be composited into its neighboring surface using virtual geometry. The options of the command allow you to specify where to split each curve. You must input a distance for the first curve (δ1), but the second location can be determined based on the length and direction of the first curve.

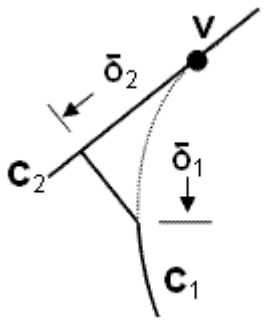


Figure 1. Collapse angle syntax

The **arclength** option will split each curve at a specified distance δ1 and δ2, (See Figure 1) measured from the vertex. You must input at least one **arclength** for each of the options listed below.

The **same_size** option will split curve 2 so that the two resulting curves, δ1 and δ2, are the same length as shown in Figure 2.

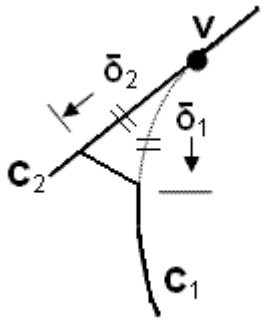


Figure 2. Collapse angle using the same_size option

The **perpendicular** option will split curve 2 so it is perpendicular to the split location on curve 1, as shown in Figure 3.

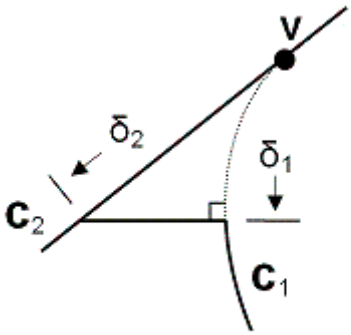


Figure 3. Collapse angle using the perpendicular option

The **tangent** option will split curve 2 where a line tangent to curve 1 at the split location intersects curve 2, as shown in Figure 4.

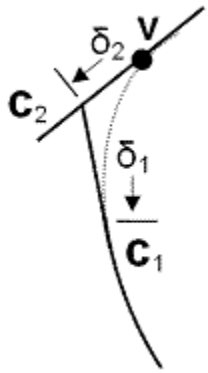


Figure 4. Collapse angle using the tangent option

The **composite_vertex** option automatically composites resulting surfaces if there are only two curves left at the vertex, and the angle is less than a specified tolerance.

The **preview** option will preview composited surface before applying changes.

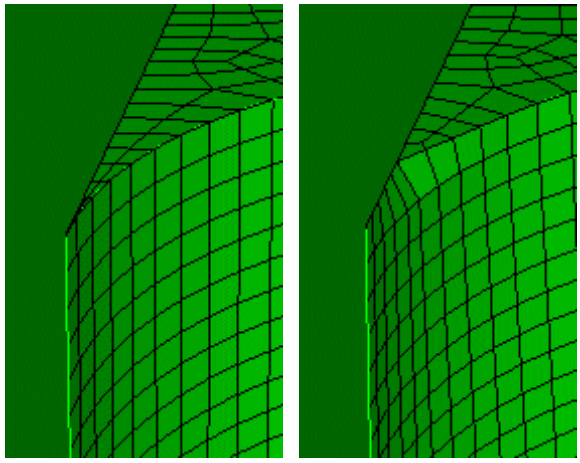


Figure 5. An example of a meshed surface that is generated after using the collapse angle command.

Deleting Virtual Geometry

Removing Virtual Geometry

The following command removes all lower-order virtual geometry from the specified entities.

virtual remove <entity_list>

Examples:

virtual remove surface 5

Removes all composite and partition curves from surface 5.

virtual remove body all

Remove all virtual geometry from all bodies.

For removing individual virtual entities, see the sections of the documentation for each type of virtual entity:

- Composite curves
- Composite surfaces
- Partition curves
- Partition surfaces

Using The Delete Command With Composites

If the general **delete** command is invoked for a composite surface, the composite surface will be removed, and the original surfaces used to define the composite will be restored to the model. The defining surfaces are NOT also deleted. As with any other non-virtual surfaces, the **delete** command will fail if the composite has a parent volume.

To delete composite surfaces with a parent volume, the composite delete command can be used. The behavior is analogous for composite curves.

If the **delete** command is used on a volume containing a composite surface or curve, or on a surface containing a composite curve, the entire volume or surface will be deleted, including the original entities used to define the composite, as those entities are also children of the entity being deleted.

Using the Delete Command With Partitions

It is recommended that the delete command not be used with partitions, as it may break subsequent usage of the merge and delete forms of the partition command for other partitions of the same real geometry entity. However, if the delete command is used for partitions, the behavior is to delete the specified partition, and when the last partition of the real geometry is deleted, to restore the original geometry.

The **delete** command can also be used on parents of partitions. For example, a volume containing partitioned surfaces, or a surface containing partitioned curves can be deleted. In this case, the specified entity will be deleted along with all of its children, including the partition entities, and the original entities that were partitioned.

Geometry Orientation

The orientation of surface and curve geometry is the direction of the normal and tangent vectors respectively.

Each surface has a forward (or top) side. The evaluation of the surface normal at any point on the surface will return a vector at that point, orthogonal to the surface and directed towards the forward side of the surface. The mesh faces generated on each surface will have the same normal direction as their owning surface.

Each curve has a forward direction and a corresponding start and end vertex. The direction of the curve is from start to end vertex. The evaluation of the tangent vector of the curve at any point along the curve will result in a vector that is both tangent to the curve and pointing in the forward direction of the curve (towards the end vertex along the path of the curve.) The mesh edges created on each curve will be oriented in the same direction as their owning curve. The exported nodes and edges of a curve mesh will be written in the order they occur along the path of the curve.

Higher-dimension geometry has uses lower-dimension geometry with an associated sense (forward or reversed) for each lower-dimension entity. For example, a volume as a sense for each surface used to bound the volume. If the surface normal points outside the volume, then the volume uses the surface with a forward sense. If the surface normal points into the interior of the volume, the volume uses the surface with a reversed sense. Similarly a surface is bounded by a set of curves forming a loop such that the direction of the loop and the sense of each curve results in a cycle that is counter-clockwise around the surface normal.

Adjusting Orientation

By default, a surface is oriented so that its normal points OUT of the volume of which it is a part. For a merged surface (a surface which belongs to more than one volume) or a free surface (a surface that belongs to no volume, also known as a sheet body), the orientation of the surface is arbitrary. The orientation of a surface influences the orientation of any elements created on that surface. All surface elements have the same orientation as the surface on which they are created. The following commands are available to adjust the normal-direction for a surface:

Surface <id_range> Normal Opposite

Surface <id_range> Normal Volume <id>

The orientation of a surface can be flipped from its current orientation by using the "Opposite" keyword. The orientation of a merged surface can be set to point OUT of a specific volume by specifying that volume in the "Volume" keyword.

The following commands are available to adjust the tangent direction of a curve:

Curve <id_range> Tangent Opposite

Curve <id_range> Tangent {Forward|Reverse} Surface <id>

Curve <id_range> Tangent {Start|End} Vertex <id>

The first command reverses the tangent direction of the curve. The second command sets the tangent direction such that it is used by a specific surface with a specified sense. The third command sets the tangent direction of the curve such that the curve starts or ends with the specified vertex. For the latter two forms of the command, the curve must be adjacent to the specified surface or vertex.

Geometry Groups

Groups provide a powerful capability for performing operations on multiple geometric entities with minimal input. They can also serve as a means for sorting geometric entities according to various criteria. The following describes the Group operations available in CUBIT:

When a group is meshed, CUBIT will automatically perform an interval matching on all surfaces in the group (including surfaces that are a part of volumes or bodies in the group).

- Basic Group Operations
- Groups in Graphics
- Propagated Hex Groups
- Quality Groups

There are several utilities in CUBIT which use groups as a means of visualizing output. These utilities are described elsewhere, but listed here for reference:

- Webcut results
- Merged and unmerged entities
- Sweep groups
- Interval matching

Basic Group Operations

Geometry Groups

The command syntax to create or modify a group is:

Group ["name" | <id>] add <list of topology entities>

For example, the command,

Group "Exterior" add surface 1 to 2, curve 3 to 5

will create the group named Exterior consisting of the listed topological entities. Any of the commands that can be applied to the "regular" topological entities can also be applied to groups. For example, **mesh Exterior** , **list Exterior** , or **draw Exterior** . A topological entity can be removed from a group using the command:

Group ["name" | <id>] remove <entity list>

The *Xor* operation can also be performed on entities in group. Xor means if an entity is already in the group, the command will delete this entity from the group. If it is not in the group, the entity is then added to the group.

Group ["name" | <id>] xor <entity list>

Mesh Groups

Groups may also contain mesh entities. The commands for adding and removing mesh entities are analogous to those for geometric entities.

Group ["name" | <id>] add {hex|face|edge|node <id_list>}

Group ["name" | <id>] remove {hex|face|edge|node <id_list>}

Group ["name" | <id>] xor {hex|face|edge|node <id_list>}

Deleting Groups

Groups can be deleted with the following command:

Delete Group <id range> [propagate]

The option **propagate** will delete the group specified and all of its contained groups recursively.

Cleaning Out Groups

You can remove all of the entities in a group via the **cleanout** command:

Group <group_id_range> Cleanout [geometry|mesh] [propagate]

By default all entities will be removed - optionally you can cleanout just geometry or mesh entities. As in delete, the **propagate** option will cleanout the group specified and all of its contained groups recursively.

Groups in Graphics

In the GUI version of CUBIT, groups may be picked with the mouse.

When displaying a group containing hexes, only the outside skin of the hexes will be displayed.

Propagated Hex Groups

Grouping propagated hexes is a mechanism for selecting groups of hexes from a hex mesh using sweep-type criteria. For example, creating a group containing all hexes between two specified mesh faces.

- Starting on a Face
- Starting on a Surface
- Naming Convention

Note: the examples above are based on first executing these commands:

brick width 10

volume 1 size 1

mesh volume 1

Propagated Hex Group Starting on a Face

When starting on a face, the propagation method can end at a surface, end at a face or can end after the number of times the user specifies:

- Ending at a Surface
- Ending at a Face
- Number of Times
- Ending at a Surface with Multiple
- Ending at a Face with Multiple
- Number of Times with Multiple
- Ending at a Face with Direction
- Ending at a Surface with Direction
- Number of Times with Direction

Ending at a Surface

Group ['name' | <id>] Add Hex Propagate [Source] Face <id range> Target Surface <id>

Example

group 2 add hex propagate face 1 11 21 target surface 2

Result: Group 2 will be created containing 30 propagated hexes (10 layers of 3 hexes)

Ending at a Face

Group ['name' | <id>] Add Hex Propagate [Source] Face <id> Target Face <id>

Example

group 2 add hex propagate face 1 target face 1721

Result: Group 2 will be created containing 5 propagated hexes (5 layers of 1 hex)

Note: Ending at a face requires starting at one face at one time, but ending at surface allows multiple start faces

Number of Times

Group ['name' | <id>] Add Hex Propagate [Source] Face <id range> Times <number>

Example

group 2 add hex propagate face 2 times 4

Result: Group 2 will be created containing 4 propagated hexes (4 layers of 1 hex)

All of these methods, ending at surface, end at a face or number of times, can be used with the "multiple" option which will create a grandparent (top-level), parent (mid-level, contained within the grandparent) and child (bottom level, contained within the parent) groups. The child groups will contain each hex layer (specified number of layers per child group), all organized into a single parent group, which is organized underneath the group ID given to the command. Subsequent propagation commands could then be executed adding to the grandparent group, but creating a new parent and child groups. This way multiple propagation "sets" can be stored in one grandparent group, if desired.

Ending at a Surface with Multiple

Group ['name' | <id>] Add Hex Propagate [Source] Face <id> Target Surface <id> Multiple <number>

Example

group 2 add hex propagate face 1 target surface 2 multiple 1

Result: Ten groups will be created and stored with their respective ids, one for each layer of hexes. These groups will be stored in the parent group, Group 3, and Group 3 will be stored in the grand parent group, Group 2. A subsequent propagation command could be executed adding to group 2 (the grandparent), which would create a single group contained in group 2 (the parent), containing the hex layer groups (the children).

Ending at a Face with Multiple

Group ['name' | <id>] Add Hex Propagate [Source] Face <id> Target Surface <id> Multiple <number>

Example

group 2 add hex propagate face 1 target face 1721 multiple 1

Result: 5 groups will be created and stored with their respective ids, one for each layer of hexes. These groups will be stored in the parent group, Group 3, and Group 3 will be stored in the grand parent group, Group 2. A subsequent propagation command could be executed adding to group 2 (the grandparent), which would create a single group contained in group 2 (the parent), containing the hex layer groups (the children).

Number of Times with Multiple

Group ['name' | <id>] Add Hex Propagate [Source] Face <id> Times <number> Multiple <number>

Example

group 2 add hex propagate face 1 times 10 multiple

Result: Two groups will be created and stored with their respective ids, these two groups will be stored in the parent group, Group 3, and Group 3 will be stored in the grand parent group, Group 2.

If the end surface or end face is ambiguous, a node direction can be specified to direct the propagation. When specify the node direction, the node has to be picked such that when the hexes are propagated, the picked node lies in these propagated hexes. If that node is never reached while propagating, the direction is not found and zero hexes will be included in the specified group.

Ending at Face with Direction

Group ['name' | <id>] Add Hex Propagate [source] Face <id> Target Face <id> Direction Node <id>

Example

group 2 add hex propagate face 1721 target face 1 direction node334

Result:group 2 will be created containing 6 hexes

Ending at Surface with Direction

Group ['name' | <id>] Add Hex Propagate [Source] Face <id range> Target Surface <id> Direction Node <id>

Example

group 2 add hex propagate face 1 target surface 2 direction node 334

Result: group 2 will be created containing 10 hexes

Note: The direction command and the multiple command can be used together (i.e. group 2 add propagate face 1721 end face 1 multiple 2 direction node 334)

If number of times is specified and the direction is ambiguous, a surface direction or a node direction can be specified to direct the propagation. The node direction has the same condition as when ending at a surface or face and that is it must lie in the propagated hexes.

Number of Times with Direction

Group ['name' | <id>] Add Hex Propagate [Source] Face <id> Times <number>Direction [surface <id> | node <id>]

Example

group 2 add hex propagate face 110 times 4 direction surface 2

group 2 add hex propagate face 1 times 4 direction node 269

Result: group 2 will be created contained 4 hexes

Note: The direction command and the multiple command can be used together. (i.e. group 2 add propagate face 1721 times 4 multiple 2 direction surface 1)

Propagated Hex Group Starting on a Surface

Starting on a surface can end at a surface or can end after the number of times the user specifies.

- Ending at a Surface
- Number of Times
- Ending at a Surface with Multiple
- Number of Times with Multiple
- Ending at a Surface with Direction
- Number of Times with Direction

Ending at a Surface

Group ['name' | <id>] Add Hex Propagate Surface <id> Target Surface <id>

Example

group 2 add hex propagate surface 1 target surface

Result: Group 2 will be created containing 1000 hexes

Number of Times

Group ['name' | <id>] Add Hex Propagate Surface <id> Times <number>

Example

group 2 add hex propagate surface 1 times 4

Result: Group 2 will be created containing 400 hexes

Both methods, ending at surface or number of times, can be used with the "multiple" option which will create several groups depending upon the multiple number specified.

Ending at a Surface with Multiple

Group ['name' | <id>] Add Hex Propagate Surface <id> Target Surface <id> Multiple <number>

Example

group 2 add hex propagate surface 1 target surface 2 multiple 2

Result: Five groups will be created and stored with their respective ids of multiple 2, these groups will be stored in the parent group, Group 3, and Group 3 will be stored in the grand parent group, Group 2.

Number of Times with Multiple

Group ['name' | <id>] Add Hex Propagate Surface <id> Times <number> Multiple <number>

Example

group 2 add hex propagate surface 1 times 10 multiple 5

Result: Two groups will be created and stored with their respective ids of multiple 5, these two groups will be stored in the parent group, Group 3, and Group 3 will be stored in the grand parent group, Group 2.

If number of times is specified and the direction is ambiguous, the surface direction or the node direction can be specified to direct the propagation. If the end surface is specified, only a node direction can be specified to direct the propagation. When specifying the node direction, the node has to be picked such that when the hexes are propagated, the picked node lies in these propagated hexes. If that node is never reached while propagating, the direction is not found and zero hexes will be included in the specified group.

Note: for the examples below, the result can be seen by executing these commands:

brick x 10
vol 1 size 1
brick width 10
body 2 move 10
volume all size 1
merge all
mesh volume all

Ending at Surface with Direction

Group ['name' | <id>] Add Hex Propagate Surface <id> Times <number> Direction Node <id>

Example

group 2 add hex propagate surface 6 target surface 12 direction node 1530

Result: Group 2 will be created containing 400 hexes

Note: The direction command and the multiple command can be combined (i.e. group 2 add propagate surface 6 times 4 multiple 2 direction node 1530)

Number of Times with Direction

Group ['name' | <id>] Add Hex Propagate Surface <id> Times <number> Direction [surface <id> | node <id>]

Example

group 2 add hex propagate surface 6 times 4 direction surface 4

group 2 add hex propagate surface 6 times 4 direction node 1530

Result: group 2 will be created containing 400 hexes

Naming Convention for Propagated Hex Groups

A special naming convention can be used for the propagated groups, best described by an example.

The following command will create a hierarchy of logically named groups, as follows.

group 'W1P1T1' add propagate surf 1 end surf 2 multiple 1

The hierarchy looks like this:

W1

W1P1

W1P1T1

W1P1T2

W1P1T3

...

W1P1T10

Where W1P1 is contained within W1, and W1P1T1, W1P1T2, etc.. are contained within W1P1.

The software simply looks for numerical numbers in the group name and parses out the correct grandparent, parent and child names from the substrings. There must be exactly 3 substrings in the group name, each ending with an integer for the command to work properly.

A subsequent command:

group 'W1P2T1' add propagate surf 3 end surf 5 multiple 1

will add a parent group to W1, called W1P2, and the subsequent child groups:

W1

W1P1

W1P1T1

W1P1T2

W1P1T3

...

W1P1T10

W1P2

W1P2T1

W1P2T2

W1P2T3

...

W1P2T10

Quality Groups

Groups can also be formed from the hexes or faces obtained from the quality command. Each group formed using quality can be drawn with its associated quality characteristics {i.e. jacobian low .2 high .3} automatically.

group ['name'|<id>] add quality {volume | surface | group | hex | face} <id range> <metric name> [low <value> | bottom <value> | top <number> | bottom <number> | malformed]

The following example illustrates the use of quality groups:

group 2 add quality volume 1 jacobian

In this case, if the meshed brick from the section Propagated Hex Groups is used, Group 2 will be created and it will contain 1000 hexes with quality characteristics.

Geometry Attributes

Each geometric topological entity has specific information attached to it. These attributes specify aspects of the entity such as the color that entity is drawn in and the meshing scheme to be used when meshing that entity. This section describes those geometry attributes that are not described elsewhere in this manual.

- Entity Names
- Entity IDs
- Persistent Attributes
- Geometry Metadata

Entity Names

By default, geometric entities in CUBIT are referenced using an entity type (e.g. Surface, Volume) and an id, for example "draw surface 1". However, geometric entities can also be assigned names, to simplify working with specific entities. Once a name is assigned to an entity, that name can be used in any CUBIT command in place of the entity type and number. For example, if surface 1 were named 'mysurf1', the command above would be equivalent to "draw mysurf1". Also, since entity names are saved with the geometry, this also provides a means for persistent identifiers for geometric entities. The following command assigns names to geometric entities in CUBIT:

{Group|Body|Volume|Surface|Curve|Vertex} Name `<entity_name>'

The name of each topological entity appears in the output of the List command. In addition, topological entities can be labeled with their names (see label command). A list of all names currently assigned and their corresponding entity type and id (optionally filtered by entity type) can be obtained with the command

list names [{group|body|volume|surface|curve|vertex|all}]

Notes:

- In a merge operation, the names of the deleted entity will be appended to the names of the surviving entity.
- A geometric entity may have multiple names, but a particular name may only refer to a single entity.

Automatic Name Creation

CUBIT provides an option for automatically assigning names to entities upon entity creation. This option is controlled with the command:

set default names {on|off}

When this option is on, entities are assigned default names consisting of a geometry type concatenated with the entity id, for example 'cur1', 'surf26', or 'vol62'.

Automatic Name Propagation

CUBIT automatically propagates names through webcuts. If an entity that has been assigned the name "Gear" is split through webcuts, the resulting bodies are named "Gear" and "Gear@A". Try the following example.

br x 10
volume 1 name "Cube"
webcut volume 1 xplane
webcut volume 1 2 yplane
webcut volume 1 2 3 4 zplane
label volume name

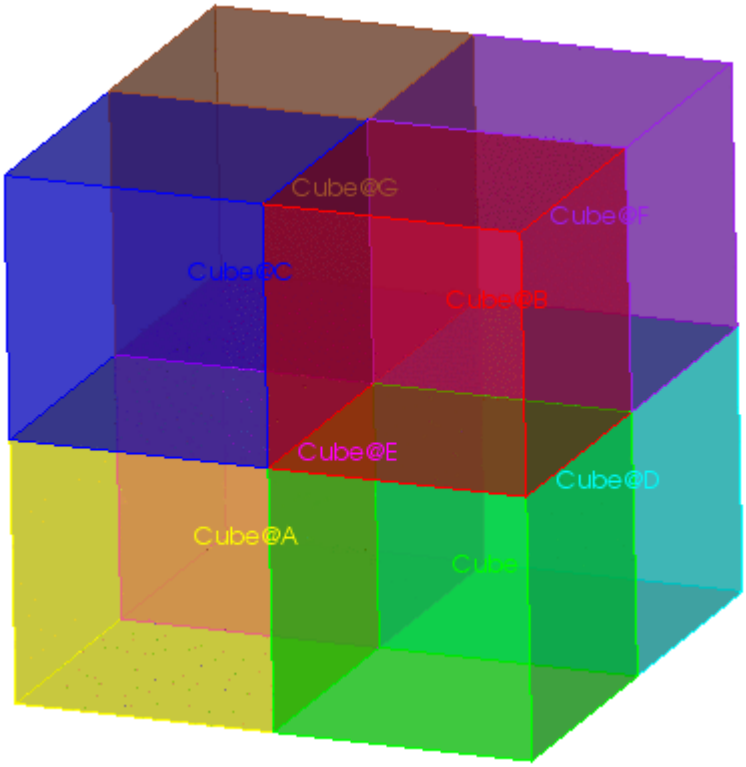


Figure 1. Name Propagation through Webcuts

Entity IDs

Topological entities (including groups) are assigned integer identification numbers or ids in CUBIT in ascending order, starting with 1 (one). Each new entity created within CUBIT receives a unique id within the topological entity type. This id can be used for specifying the entity in CUBIT commands, for example "draw volume 3".

Gaps in ID space

After working with a geometry model for some time, various operations will cause gaps to be left in the numbering of the geometric entities. The **compress ids** command can be used to eliminate these gaps:

Compress [ids] [all] [group | body | volume | surface | curve | vertex | hex | face | edge | node]
[retainmax] [sort]

Typing **compress** with no options or **compress all** will compress the ids of all entities; otherwise, the entity type for which ids should be compressed can be specified.The **retainmax** argument will retain the maximum id for each entity type, so that entities created subsequent to this command will receive ids greater than that value. If the **Sort** qualifier is included, the new id of each entity will be determined by its size and location. Small entities are given a lower id than large entities. Entities that are the same size are sorted by their location, with lower x, y, and z coordinates leading to a lower id. If two entities are found to have the same size and location, they are sorted according to their previous ids. This option can be used to restore ids in translated models in a manner which leads to more persistence than purely random id assignment.

Persistent Attributes

Typical data assigned to topological entities during a meshing session might include intervals, mesh schemes, group assignments, etc. By default, most of this data is lost between CUBIT sessions, and must be restored using the original CUBIT commands. Using CUBIT's persistent attributes capability, some of this data can be saved with the solid model and restored automatically when the model is imported into CUBIT.

- Attribute Behavior
- Attribute Types
- Attribute Commands
- Using CUBIT Attributes

Attribute Behavior

In this context, attributes are defined as data associated directly with a particular geometry entity. In CUBIT's implementation of attributes, these data can occupy one of three "states" at any given time: they can be stored in data fields on CUBIT's geometry entities; they can be stored in an intermediate representation, using CUBIT's attribute objects; or they can exist only on the ACIS objects. When they are stored on ACIS objects, those attributes are written to and read from disk files with the geometry. This mechanism allows CUBIT-specific information to be stored and retrieved with the geometry data. By default, attribute data is not stored with geometry. To enable the use of attributes, use the commands described in the following sections.

Attribute Types

The attribute types currently implemented in CUBIT are shown below.

Attribute Types	Description
Color	Entity Color
Composite vg	Used to restore composite virtual topology
Genesis entity	Membership in boundary conditions (block, sideset, nodeset)
Id	Entity Id
Mesh container	Handle to mesh defined for the owner
Mesh scheme	Meshing scheme (e.g. paving, sweeping, etc.)
Name	Entity name

Partition vg	Used to restore partition virtual topology
Smooth scheme	Smoothing scheme (e.g. Laplacian, Condition Number)
Unique Id	Unique entity id, used to cross-reference other entities
Vertex type	Used to define mesh topology at vertex for mapping/submapping
Virtual vg	Used to store virtual geometry entity(ies) defined on an entity

Attribute Commands

Most non-CUBIT-developer uses of attributes will be to use all or none of the attributes. Therefore, the most common command to enable and disable the use of attributes is:

Set Attribute {on | off}

When this option is on, all defined attributes will be saved with the geometry when the user enters the Export Acis command.

When a geometry is imported into CUBIT, any attributes defined on that geometry and recognized as CUBIT attributes are imported and put into an intermediate representation (that is, this information is not assigned directly to the geometry entities). To find out which attributes are defined on a given set of entities, use the following command:

List [<entity_list>] Attributes [Type <attribute type>] [All] [Print]

If no entities are entered, attribute information for all the geometric entities defined in CUBIT is printed.

The **Type** option can be used to list information about a specific attribute type; values for are the same as those in the previous table.

If the **All** option is entered, information about all attribute types will be printed, even if there are none of those attributes defined for the specified entities.

If the **Print** option is entered, the information stored in each attribute will be printed; this command is usually used only by CUBIT developers.

Control By Attribute Type or Geometric Entity

Attributes can be enabled or disabled by attribute type, to allow the use of only user-specified attribute types. To turn on or off specific attributes, use the command:

Set Attribute <attribute type> {on | off}

where **<attribute type>** is one of the types shown in the previous table.

Attributes can also be controlled to automatically write (update) and read (actuate) to/from solid model files automatically, using the command:

Set Attribute <attribute_type> Auto {Actuate | Update} {on | off}

Finally, attributes can be manually written to and read from the geometric entities, and removed from cubit entities, using the command

{geom_list} Attribute {all | attribute_type} {actuate | remove | update | read | write}

where **geom_list** is a list of geometry entities. This command is recommended only for developers' use.

Using CUBIT Attributes

A typical scenario for using CUBIT attributes would be as follows.

Construct geometry, merge, assign intervals, groups, etc. (i.e. normal CUBIT session)

Enable automatic use of attributes using the command:

Set Attribute On

Export acis file (see Export Acis command).

Subsequent runs:

Enable automatic reading and actuating of attributes:

Set Attribute on

Import ACIS file (see Import Acis command)

Used in this manner, geometry attributes allow the user to store some data directly with the geometry, and have that data be assigned to the corresponding CUBIT objects without entering any additional commands.

Geometry Metadata

Geometry Metadata enables CUBIT to preserve assembly relationships and material data through the analysis process. Metadata may include a model's assembly structure, or the division of volumes into assemblies, sub-assemblies, and parts. Metadata may also include additional data such as part numbers and material descriptions. CUBIT also generates metadata which describes the relationship between element blocks and assembly parts.

CUBIT's metadata capabilities are part of the material transparency effort of the DART project (see the DART project's Analyst Home Page at http://www-irn.sandia.gov/analyst).

This page describes the procedures for importing, manipulating and exporting Metadata within CUBIT.

- Assembly Structure
- Viewing Metadata
- Modifying Metadata
- Viewing and Modifying Global Metadata
- Removing Metadata Associations
- Replacing Metadata on Imported Geometry
- Exporting Metadata

Assembly Structure

Volumes can be organized into parts, assemblies, and sub-assemblies. There are two ways to create an assembly structure. The first is to import an XML file along with a STEP file, using the XML option of the import step command.

Import step "<step_filename>" [XML "<xml_filename>"]

The XML file must follow the DART Metadata XML schema found at http://www-irn.sandia.gov/~mfcurre/metadata/DARTMetadata_final.xsd and described at http://www-irn.sandia.gov/~mfcurre/metadata/DARTXMLschema.htm (This website is subject to change -- The final website will be accessible from somewhere under the Analyst Home Page). For the XML data to be correctly imported, the STEP file must also contain extra data, namely an attribute on each volume that corresponds the base filename of the file href for a given part in the XML (base filename has no path and no extension). Both the XML file and the STEP file can be exported in their correct formats from Pro/E using an add-in called eMatrix (another tool under the umbrella of the DART project, see the Analyst Home Page for details).

The second way to create the assembly tree is to modify the "name" metadata of a volume.

Metadata Modify Name 'new value' Volume <id>

When you modify a volume's metadata name, a part is created for that volume. That volume is placed as a top-level part in the assembly tree such that its path is "/name". At this time you can't organize cubit-created parts into assemblies and sub-assemblies, but this will be possible at some time in the future.

Viewing Metadata

Once an assembly tree is created the metadata can be viewed using the following command

Metadata List [Name | Description | Material_Description | Material_Specification | File | Units | Instance | ALL] {Volume | Part 'PartPath'[instance]}

If you list a part, you must include its full path. Since it is possible for more than one part to have the same path, you can include an instance number to indicate which instance you are interested in. If you don't include an instance number, CUBIT will list the first part it finds with a matching path.

Modifying Metadata

Existing metadata can be modified using the modify command. The full command syntax is:

Metadata Modify { Name | Description | Material_Description | Material_Specification | File | Units | Instance} 'new value' {Volume|Part 'PartPath'[instance]}

The specified piece of metadata will be changed to "new_value". Note that modifying metadata for a volume that isn't associated with a part will create a new part for that volume, as described above. Also note that metadata is tracked through geometry operations. Take the example where volume 1 is associated with a part called /path/part. If you webcut volume 1 into volume 1 and 2, then both of the new volumes are now associated with /path/part.

Viewing and Modifying Global Metadata

There is also metadata that applies to the whole assembly tree, not a single part. These metadata can be viewed and changed using the following commands:

Metadata List {Classification_Level | Classification_Category | Weapon_Category}

Metadata Modify {Classification_Level | Classification_Category | Weapon_Category} 'new_value' {Volume | Part 'PartPath'[instance]}

Removing Metadata Associations

You can remove the association between a part and a volume using the command:

Metadata Remove {Volume | Part 'PartPath'[instance]}

This command removes the volume from the part. After this command is issued, the volume no longer belongs to any part and has no metadata. You can also remove a whole part at a time, as in "metadata remove part '/path/to/my/part/part_name". This will remove the metadata for all volumes associated with the part.

Replacing Metadata on Imported Geometry

Sometimes you want to replace imported geometry with volumes you created in CUBIT. An example of when you would want to do this is when you are simplifying geometry, or when you are replacing "bad" geometry with good geometry. The volumes in the first list will be removed from their part, and the volumes in the second list will be added to the part. Note that all volumes in the first list must belong to the same part. Otherwise you get an error since CUBIT can't determine which part to add the new volumes to.

Metadata Replace {Volume} <id> with {Volume} <id>

The command will remove all volumes in the first volume list from their part, and then add the second list of volumes to that part.

Exporting Metadata

When you export a mesh and include the XML option, a genesis file is written out in the same format as always, and an XML file is written out as well. This XML file follows the DART schema, and includes a mesh section which provides mapping between element blocks in the genesis file and parts in the metadata. See the Analyst Home Page for details.

Export Mesh "file.g" [XML "file.xml"]

Geometry Deletion

Geometry can be deleted from the model using the following command:

Delete [Body | Surface | Curve | Vertex] <id_range>

Any type of Body can be deleted, whether it is based on solid model geometry or another representation. Other entities (Surface, Curve, Vertex) can be deleted when they are "free", i.e. when they are not contained in an entity of higher topological order (Body, Surface or Curve, respectively); this type of geometry is often created from the lowest order topology up.

Mesh Generation

- Interval Assignment
- Meshing Schemes
- Meshing the Geometry
- Mesh Quality Assessment
- Mesh Modification
- Mesh Validity
- Mesh Adaptivity and Sizing Functions
- Mesh Deletion

The methods used to generate a mesh on existing geometry are discussed in this chapter. The definitions used to describe the process are first presented, followed by descriptions of interval specification, mesh scheme selection, and available curve, surface, and volume meshing techniques. The chapter concludes with a description of the mesh editing capabilities, and the quality metrics available for viewing mesh quality.

Element Types

For each entity topology-type in the model geometry, CUBIT can discretize the entity using one, or several, types of basic elements, for each order entity in the geometry (vertex, curve, etc.). CUBIT uses a basic element designator to describe the corresponding entity, or entities, in the mesh, and a given geometric topology entity can be discretized with one, or several, of basic elements types in CUBIT. For example, a geometric surface in CUBIT is discretized into a number of faces, where faces is the basic element designator for surfaces. These faces can consist of two types of basic elements, quadrilaterals or triangles. The basic element designators corresponding to each type of geometric entity, along with the types of basic elements supported in CUBIT, are summarized in the table below.

Geometry Entity Type	Basic Element Designator	Basic Element(s) In CUBIT
Vertex	Node	Node
Curve	Edge	Edge
Surface	Face	Quadrilateral, Triangle
Volume (or Body)	Element	Hexahedron, Tetrahedron, Pyramid

For each basic element, CUBIT also supports several element type definitions, whose use depends on the level of accuracy desired in the finite element analysis. For example, CUBIT can write both linear (4-noded) and quadratic (8- or 9-noded) quadrilaterals. The element type definition is specified after meshing occurs, as part of the boundary condition specification. See Finite Element Model Definition for a description of that process and the various element types available in CUBIT.

Each mesh entity is associated with a geometric entity which "owns" it. This associativity allows the user to mesh, display, color, and attach attributes to the mesh through the geometry. For example, setting a mesh attribute on a surface affects all faces owned by that surface.

Mesh Generation Process

Starting with a geometric model, the mesh generation process in CUBIT consists of four primary steps:

Set interval size and count for individual entities or groups

The size or interval is always applied to a specific geometric entity. For example:

volume 1 size 2.0

Set mesh schemes

CUBIT supports numerous meshing schemes for meshing solid model entities. For example:

volume 1 scheme sweep

Generate the mesh for the model

Use the mesh command to generate the mesh on a specified geometric entity. For example:

mesh volume 1

Inspect mesh for quality and suitability for targeted analysis

CUBIT provides various quality metrics for the user to verify the suitability of the mesh for analysis. The quality command can be used to check the elements generated on a specific geometric entity. For example:

quality volume 1

There are also mechanisms for improving mesh quality locally using smoothing and local mesh topology changes and refinement. For complex models, this process can be iterative, repeating all of the steps above.

The mesh for any given geometry is usually generated hierarchically. For example, if the mesh command is issued on a volume, first its vertices are meshed with nodes, then curves are meshed with edges, then surfaces are meshed with faces, and finally the volume is meshed with hexes. Vertex meshing is of course trivial and thus the user is given little control over this process. However, curve, surface, and volume meshing can be directly controlled by the user. Each of the steps listed are described in detail in the following sections.

Interval Assignment

- Interval Firmness
- Explicit Specification of Intervals
- Automatic Specification of Intervals
- Interval Matching
- Periodic Intervals
- Relative Intervals
- Mesh Preview

Mesh density is usually controlled by the intervals, i.e. the number of mesh edges, specified on curves. Intervals are set either directly by specifying the interval count for a curve, or by specifying a desired size for each interval on a curve. Intervals can be specified for curves individually, or indirectly by specifying intervals for higher order geometry containing those curves. Because of interval constraints imposed by various meshing algorithms in CUBIT, the assignment of intervals to curves is not completely arbitrary. For this reason, a global interval match must be performed prior to meshing one or more surfaces or volumes.

Interval Firmness

Before describing the methods used to set and change intervals, it is important that the user understand the concept of interval firmness. An interval firmness value is assigned to a geometry curve along with an interval count or size; this firmness is one of the following values:

hard: interval count is fixed and is not adjusted by interval size command or by interval matching

soft: current interval count is a goal and may be adjusted up or down slightly by interval matching or changed by other interval size commands.

default: default firmness setting, used for detecting whether intervals have been set explicitly by the user or by other tools

Interval firmness is used in several ways in CUBIT. Each curve is assigned an interval firmness along with an interval count or size. Commands and tools which change intervals also affect the interval firmness of the curves. Those same commands and tools which change intervals can only do so if the curves being changed have a lower-precedence interval firmness. The firmness settings are listed above in order of decreasing precedence. For example, some commands are only able to change curves whose interval firmness is soft or default ; curves with hard firmness are not changed by these commands.

More examples of interval setting commands and how they are affected by firmness are given in the following sections.

A curve's interval firmness can be set explicitly by the user, either for an individual curve or for all the curves contained in a higher order entity, using the command:

{geom_list} Interval {Default | Soft | Hard}

All curves are initialized with an interval firmness of default , and any command that changes intervals (including interval assignment) upgrades the firmness to at least soft .

Precedence

If a size is specified multiple times for a single entity, the following precedence is used:

- The highest firmness command takes precedence.
  Hard commands include "curve <id> interval <val>", and "{geometry_list} interval hard" will fix the size at the current size.
- Within a given firmness, the last-issued command takes precedence.
  For example, if the user commands "surface 1 size 1" then "volume 1 size 2", and surface 1 is part of volume 1, then surface 1 will have a size of 2.

Explicit Specification of Intervals

The density of edges along curves is specified by setting the actual number of intervals or by specifying a desired interval size. The number of intervals or interval size can be explicitly set curve by curve, or implicitly set by specifying the intervals or interval size on a surface or volume containing that edge. For example, setting the intervals for a volume sets the intervals on all curves in that volume.

The commands to specify the number of intervals at the command line are:

{Curve|Surface|Volume|Body|Group} <range> Interval <intervals>

{Curve|Surface|Volume|Body|Group} <range> [Interval] Size <interval_size>

The first command above sets interval counts. When setting interval counts for surfaces, volumes, bodies and groups, an intervals firmness of soft is assigned to the owned curves. When setting the interval count for a curve, a firmness of hard is assigned.

Interval size may be specified as well; the interval count for each owned curve is computed by dividing the curve's arc length by the specified interval size. Interval size commands always assign a firmness of soft to the specified entities.

The user can scale the current intervals or size with the following commands. Scaling is done on an entity by entity basis.

{Curve|Surface|Volume|Body|Group} <range> Interval Factor <factor>

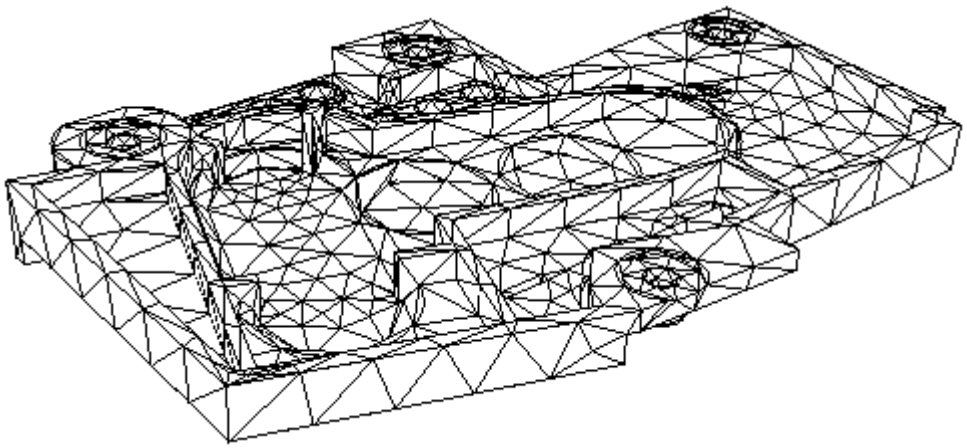
{Curve|Surface|Volume|Body|Group} <range> [Interval] Size Factor <factor>

Automatic Specification of Intervals

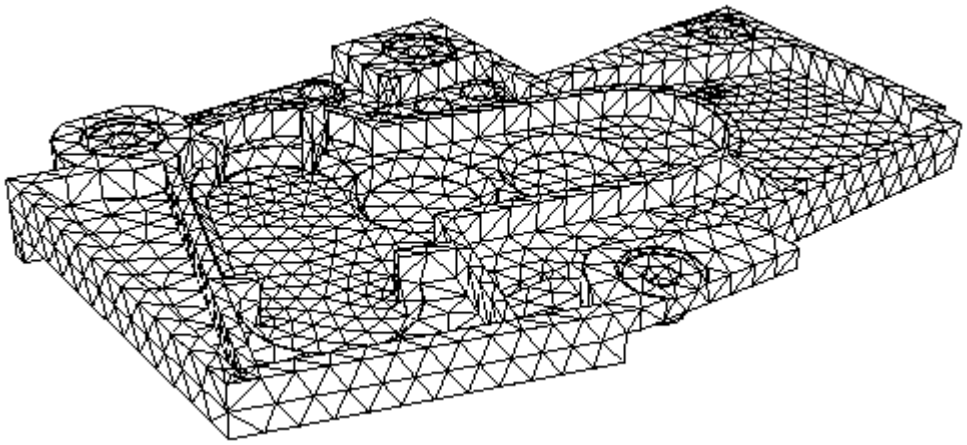
In addition to specifying intervals explicitly based on a known count or size, CUBIT is also able to compute interval counts automatically based on characteristics of the model geometry. The following automatic interval setting command can be used:

{geom_list} size auto [factor <factor>]

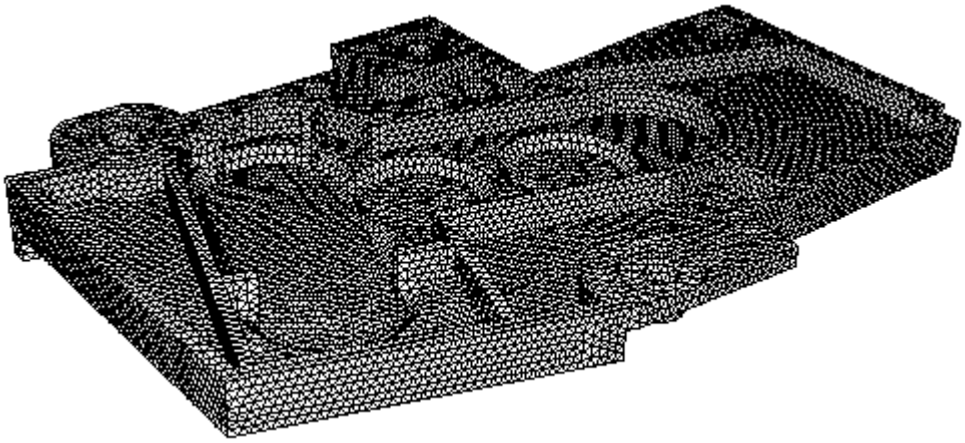
Vertices are not valid in the geom_list for this command. Automatic interval assignment works by examining the geometric characteristics of the entities in the geom_list and assigning a heuristic size to the entities and their child entities. The factor may be a floating point number between 1 and 10, where 1 represents a fine interval size and 10 represents a coarse size. Figure 1 shows an example of different auto size specification on a CAD model.



(a) auto size factor = 7.0



(b) auto size factor = 5.0



(c) auto size factor = 1.0

The user may assign the interval size to be the arc length of the smallest curve contained in the specified entity or entities using the following command:

{geom_list} size smallest curve

Vertices are not allowed in the geom_list for this command. This command assigns a soft interval firmness.

Default auto interval specification

If intervals have not been explicitly defined by the user for the curves or their owning surfaces and volumes, an **auto size factor** of **5** will automatically be computed for the entities being meshed. The automatic size specifications can be overridden easily by specifying another **auto size factor** or an explicit interval size.

If an **auto size factor** of **5** is undesirable for most meshing operations, the default factor may be changed by using the following command:

set auto size default <value>

where value is a number from **1** to **10**. This will be the default auto size factor used when either a factor has not been specified on the **size auto** command or the entity is meshed without otherwise setting explicit intervals or size.

In previous versions of CUBIT a default interval of 1 was assigned to all entities. If this behavior is still desired, the following command may be used to enforce this condition:

set default autosize [ON|off]

Maximum Spanning Angle on Arcs

On many CAD models, arcs or small holes require that a finer mesh be specified around these entities in order to maintain reasonable mesh quality. To facilitate this, the user may specify the maximum angle an element edge may span on an arc. To change or list the maximum arc span, use the following commands

set maximum arc_span <angle>

list maximum arc_span

Where angle is a positive value less than 360. The maximum arc span setting will only be used if there is not already a user defined interval set on the arc. Figure 2 shows the effect of three different maximum arc_span settings on a simple cylinder using the sweep mesh scheme.

Figure 2. maximum arc_span settings of 90, 45 and 15 degrees respectively.

Default arc span setting: In addition to setting an automatic size factor, if there are otherwise no user-defined interval sizes defined on an arc and no **maximum arc_span** has been set by the user when a tetrahedral mesh or triangle mesh is defined, a maximum spanning angle of **60 degrees** will be used.

Note that once interval sizes have been defined when the entity has been meshed, it may be necessary to reset the interval settings **(reset {geom_list}**) to use a new maximum arc span setting when remeshing.

Interval Matching

Each meshing scheme in CUBIT imposes a set of constraints on the intervals assigned to the curves bounding the entity being meshed. For example, meshing any surface with quadrilaterals requires that the surface be bounded by an even number of mesh edges. This constrains the intervals on the bounding curves to sum to an even number. For a collection of connected surfaces and volumes, these interval constraints must be resolved globally to ensure that each surface will be meshable with the assigned scheme. The global solution technique implemented in CUBIT is referred to as interval matching.

When meshing a surface or volume, matching intervals is performed automatically. In some cases, interval matching needs to be invoked manually, for example when meshing a collection of volumes, or a collection of surfaces not in a common volume. Interval matching can also be called to check whether the assigned intervals and schemes are compatible.

The command syntax for manually matching intervals is the following:

Match Intervals {Surface|Volume|Body|Group} <range>

Here the entity list can be any mixed collection of groups, bodies, volumes, surfaces and curves.

The interval matcher assigns intervals as close as possible to the user-specified intervals, while satisfying global interval constraints. The goal is to minimize the relative change in pre-assigned intervals on all entities. Interval matching only changes curves with interval firmness of soft or default .

Extra constraints can be added by the user to improve mesh quality locally; in particular, curves can be constrained to have the same intervals using the command

Curve <range> Interval {Same|Different}

Specifying that curves have the "same" intervals stores them in a set. More curves may be added to an existing set, and sets merged, by future commands. The current contents of the affected sets are printed after each command. A curve may be removed from a set by specifying that its intervals are "different."

The interval assignment algorithm tries to find one good interval solution from among the possibly infinite set of solutions. However, if many curves are hard-set or already meshed, there may be no solution. To improve the chances of finding a solution, it is suggested that curves are soft-set whenever possible. Also, a solution might not exist due to the way the local selections of corners and sides of mapped surfaces interact globally. If there is no solution, the following command may help in determining the cause:

Match Intervals {Surface|Volume|Body|Group} <range> [Seed Curve <range>] [Assign Groups [Only|Infeasible]] [Map|Pave]

Specifying Assign Groups will create groups that contain independent subproblems of the global problem. Specifying Assign Groups Only will group independent subproblems, but the algorithm will not attempt to solve these subproblems. Assign Groups Infeasible will put each independent subproblem with no solution into specially named groups. Often poor corner choices and surface meshing schemes will be illuminated this way. If Map or Pave is specified, then only subproblems involving mapping or paving constraints will be considered. If a Seed Curve is specified, then only those subproblems containing that curve will be considered.

Advanced users may also wish to experiment with setting the following, which may change the interval solution slightly:

Set Match Intervals Rounding {on|off}

Set Match Intervals Fast {on|off}

The user can also constrain the parity of intervals on curves:

{Curve|Surface|Volume} <range> Interval {Even | Odd}

If **Even** is specified, then during subsequent interval setting commands and during interval assignment, curves are forced to have an even number of intervals. If the current number of intervals is odd, then it is increased by one to be even. If **Odd** is specified then intervals may be either even or odd. Setting intervals to even is useful in problems where adjoining faces are paved one by one without global interval assignment.

Periodic Intervals

The number of intervals on a periodic surface, such as a cylinder, in the dimension that is not represented by a curve is usually set implicitly by the surface size.

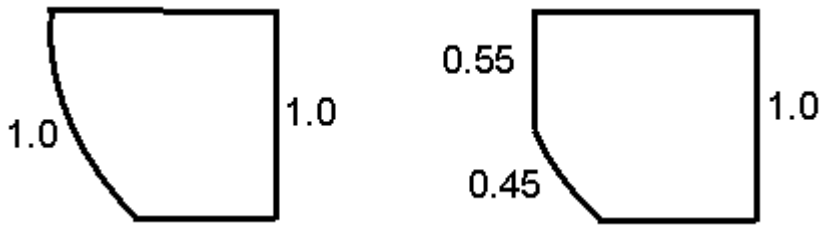
However, periodic intervals and firmness can be specified explicitly by the following commands:

Surface <range> Periodic Interval <intervals>

Surface <range> Periodic Interval {Default|Soft|Hard}

Relative Intervals

If the user needs fine control over mesh density, then for curvy or slanted sides of swept geometries, it is often useful to treat curves as if they had a different length when setting interval sizes. For example, the user may wish to specify that a slanting side curve and a straight side curve have the same "relative" length, despite their true length as shown in the following figure. These are not interval matching constraints; interval matching may change intervals so that the user-specified ratio does not hold exactly.



The relative lengths of curves are set with the following command:

{geom_list} Relative Length <size>

The following command is used to assign intervals proportional to these lengths:

{geom_list} Relative Interval <base_interval>

For a curve with relative length x, setting a relative interval of y produces xy intervals, rounded to the nearest integer.

Mesh Interval Preview

It is sometimes useful to view the nodal locations/intervals on curves graphically before meshing (which can take considerably more time). The command to do this is:

Preview Mesh {body|volume|surface|curve|vertex} <id_range> [hard]

To clear the display of the temporary nodes, simply issue a "**display**" command. The purpose of the **hard** option is that only curves that have an interval firmness of hard will be previewed.

Meshing Schemes

Meshing schemes in CUBIT can be divided into four broad categories.

- Traditional Meshing Schemes
- Free Meshing Schemes
- Conversional Meshing Schemes
- Duplication Meshing Schemes

Traditional Meshing Schemes

Traditional meshing schemes are used to apply a mesh to an existing geometry using the methods described in Meshing the Geometry (i.e. setting a scheme, applying interval sizes, and meshing). Traditional meshing schemes are available for all geometry types.

- Bias, Dualbias
- Circle
- Curvature
- Equal
- Hole
- Mapping
- Pave
- Pentagon
- Pinpoint
- Polyhedron
- Sphere
- STransition
- Stretch
- Submap
- Sweep
- Tetmesh
- Tetprimitive
- Tridelaunay
- Trimap
- Trimesh
- Tripave
- Triprimitive

Free Meshing Schemes

Free meshing schemes will create a free-standing mesh without any existing geometry

- Radialmesh

Conversional Meshing Schemes

Conversional meshing schemes are used to convert an existing mesh into a mesh of different element type or size. For example, the THex scheme will convert a tetrahedral mesh into a hexahedral mesh.

- Dice
- HTet
- QTri

- THex

Duplication Meshing Schemes

Duplication meshing schemes are used to copy an existing mesh from one geometry onto another similar geometry.

- Copy
- Mirror

General Meshing Information

Information on specific mesh schemes available in CUBIT is given in this section. The following sections have important meshing-related information as well, and should be read before applying any of the mesh schemes described below.

In most cases, meshing a geometric entity in CUBIT consists of three steps:

- Set the interval number or size for the entity (See Interval Assignment.)
- Set the scheme for the object, along with any scheme-specific information, using the scheme setting commands described below.
- Mesh the object, using the command:

  Mesh {geom_list}

This command will match intervals on the given entity, then mesh any unmeshed lower order entities, then mesh the given entity.

After meshing is completed, the mesh quality is automatically checked (see Mesh Quality Assessment), then the mesh is drawn in the graphics window.

The following table classifies the meshing schemes with respect to their applicable geometry.

Curves	Surfaces	Volumes
Bias/Dualbias	Circle	Copy
Copy	Copy	Dice
Curvature	Dice	HTet
Dice	Hole	Mapping
Equal	Mapping	Polyhedron
Pinpoint	Mirror	Sphere
Stretch	Pave	Submap
	Pentagon	Sweep
	Polyhedron	TetMesh, TetINTRIA
	QTri	Tetprimitive
	Submap	THex

	TriDelaunay	
	Triprimitive	
	TriMap	
	TriMesh, TriAdvance	
	TriPave	
	STransition	

Bias Dualbias

Summary:Meshes a curve with node spacing biased toward one or both curve ends.

Syntax:

Curve <range> Scheme Bias {Factor|First_Delta|Fraction} <double> [Start Vertex <id>] [preview]

Curve <range> Scheme Dualbias {Factor|First_Delta|Fraction} <double> [preview]

Curve <range> Scheme Bias Fine Size <double>
{Coarse Size <double> | Factor <double>} [Start Vertex <id>] [preview]

Curve <range> Scheme Dualbias Fine Size <double>
{Coarse Size <double> | Factor <double>} [preview]

Related Commands:

Curve <range> Reverse Bias

See also surface sizing function type bias

See also curve scheme stretch

The main differences between scheme bias and stretch are the following: scheme stretch does not use strict geometric series for node placement. If you specify scheme bias or dualbias using the "fine size" form, the interval count will be hard-set to a value that fills in the curve.

Discussion:

The Bias and DualBias schemes space the curve mesh unequally, placing more nodes towards (or away from) the ends of the curve according to a geometric progression. The ratio of successive edges is the "factor," which may be greater than or less than one. For bias, the series starts at the first vertex of the curve, or the "start vertex" if specified. For dualbias, the series starts at both ends of the curve and meets in the middle.

The command behaves differently depending on which set of parameters are specified. There are three basic variables: the interval count, the bias factor, or the first edge size. The curve length is a given, fixed quantity. The user can specify any two of these variables, and the third will be automatically determined.

If the "{Factor|First_Delta|Fraction}" form is specified, then the interval count is taken as a given. The interval count is whatever was specified previously by an interval count or size command (see Interval Assignment). If "Factor" is specified, then the first edge size will be automatically chosen so that the geometric progression of edges "fit" onto the curve. If "first_delta" is specified, then the first edge length is exactly that absolute value, and the "factor" is automatically chosen. If "fraction" is specified, then the first edge length is the curve length times that fraction, and again the "factor" is automatically chosen.

If the "fine size" is specified, then the first edge length is exactly that absolute value. If the "factor" is specified, then the interval count is automatically chosen. If an approximate coarse size is specified, then this also determines the factor, and again the interval count is automatically chosen. If a surface sizing function type bias is used, then the curves of the surface are sized using similar formulas.

If no start or end vertex is specified, the curve's start vertex is used as the starting point of the bias. (A curve's start vertex can be identified by listing the curve from the "CUBIT>" prompt.)

If a curve, meshed with the bias scheme, needs to have its nodes distributed towards the opposite end, it can be easily edited using the reverse bias command. Reversing the curve bias using this command is equivalent to setting a bias factor equal to the inverse of the original bias factor.

The preview option will allow the user to preview mesh size and distribution on the curve before meshing.

The following figure shows the result of meshing edges with equal, bias and dualbias schemes.

Circle

Applies to: Surfaces

Summary: Produces a circle-primitive mesh for a surface

Syntax:

Surface <range> Scheme Circle [Interval <int> | Delta_r <double>] [fraction <double>]

Discussion:

The Circle scheme is used in regions that should be meshed as a circle. A "circle" consists of a single loop of bounding curves containing an even number of intervals. Thus, the circle scheme can be applied to circles, ellipses, ovals, and regions with "corners" (e.g. polygons). The bounding curves should enclose a convex region. Non-planar bounding loops can also be meshed using the circle primitive provided the surface curvature is not too great. The mesh resembles that obtained via polar coordinates except that the cells at the "center" are quadrilaterals, not triangles. See Figure 1 for an example of a circle mesh. Radial grading of the mesh may be achieved via the optional [intervals] input parameter or by specifying the radial size [delta_r] of the outermost element. The Fraction option has the range 0 < fraction < 1 and defaults to 0.5. Fraction determines the size of the inner portion of the circle mesh relative to the total radius of the circle.

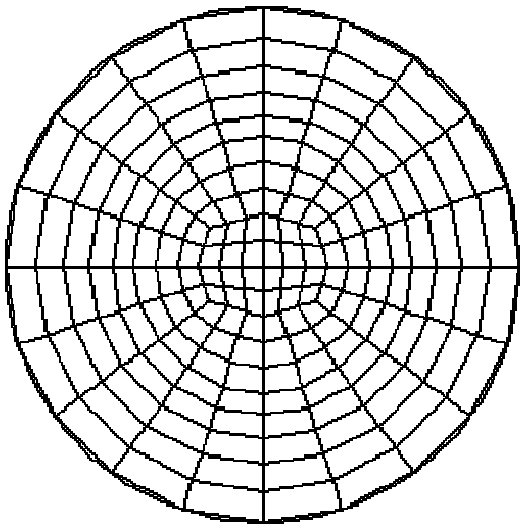


Figure 1. Circle Primitive Mesh

Curvature

Applies to: Curves

Summary: Meshes curves by adapting the interval size to the local curvature.

Syntax:

Curve <range> Scheme Curvature <double>

Discussion:

The value of <double> controls the degree of adaption. If zero, the resulting mesh will have nearly equal intervals. If greater than zero, the smallest intervals will correspond to the locations of largest curvature. If less than zero, the largest intervals will correspond to the locations of largest curvature. The default value of <double> is zero. Straight lines and circular arcs will produce meshes with near-equal intervals. The method for generating this mesh is iterative and may sometimes not converge. If the method does not converge, either the <double> is too large (over-adaption) or the number of intervals is too small. Currently, the scheme does not work on periodic curves.

Equal

Applies to: Curves

Summary: Meshes a curve with equally-spaced nodes

Syntax:

Curve <range> scheme Equal

Discussion:

See Interval Assignment for a description of how to set the number of nodes or the node spacing on a curve.

Hole

Applies to: Annular Surfaces

Summary: Useful on annular surfaces to produce a "polar coordinate" type mesh (with the singularity removed).

Syntax:

Surface <surface_id_range> Scheme Hole [Rad_intervals <int>] [Bias <double>] [Pair Node <id>
With Node <id>]

Discussion:

A polar coordinate-like mesh with the singularity removed is produced with this scheme. The azimuthal coordinate lines will be of constant radius (unlike scheme map) The number of intervals in the azimuthal direction is controlled by setting the number of intervals on the inner and outer bounding loops of the surface (the number of intervals must be the same on each loop). The number of intervals in the radial direction is controlled by the user input, rad_intervals (default is one).

A bias may be put on the mesh in the radial direction via the input parameter bias. The default bias of 0 gives a uniform grading, a bias less than zero gives smaller radial intervals near the inner loop, and a bias greater than zero gives smaller radial intervals near the outer loop.

The correspondence between mesh nodes on the inner and outer boundaries is controlled with the pair node "<loop node-id> with node <loop node-id>" construct. One id on the inner loop and one id on the outer loop should be given to connect the two nodes by a radial mesh line. Not choosing this option may result in sub-optimal node pairings with possible negative Jacobians. To use this option, mesh the inner and outer curve loops and then determine the mesh node ids.

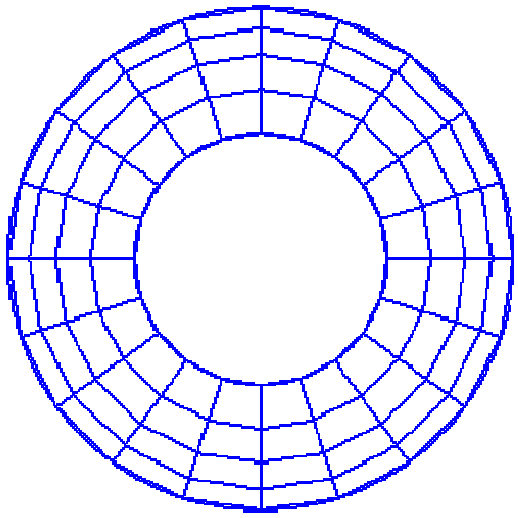


Figure 1. Example of Hole Scheme

Mapping

Applies to: Surfaces, Volumes

Summary: Meshes a surface/volume with a structured mesh of quadrilaterals/hexahedra.

Syntax:

{Volume|Surface} <range> Scheme Map

Discussion:

A structured mesh is defined as one where each interior node on a surface/volume is connected to 4/6 other nodes. Mappable surfaces contain four logical sides and four logical corners of the map; each side can be composed of one or several geometric curves. Similarly, mappable volumes have six logical sides and eight logical corners; each side can consist of one or several geometric surfaces. For example, in Figure 1 below, the logical corners selected by the algorithm are indicated by arrows. Between these vertices the logical sides are defined; these sides are described in Table 1.

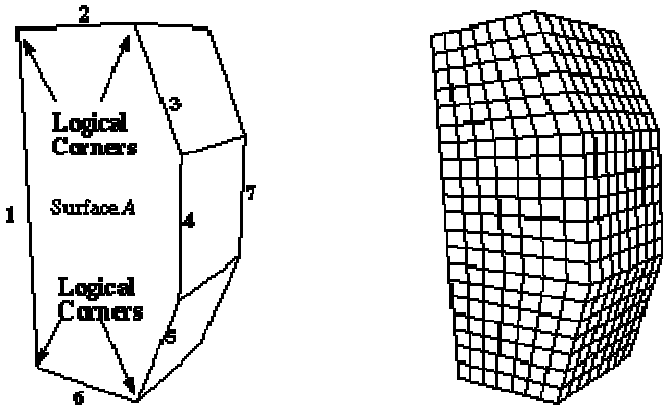


Figure 1. Scheme Map Logical Properties

Table 1. Listing of Logical Sides

Logical Side	Curve Groups
Side 1	Curve 1
Side 2	Curve 2
Side 3	Curve 3, Curve 4, Curve 5
Side 4	Curve 6

Interval divisions on opposite sides of the logical rectangle are matched to produce the mesh shown in the right portion of Figure 1. (i.e. The number of intervals on logical side 1 is equated to the number of intervals on logical side 3). The process is similar for volume mapping except that a logical hexahedron is formed from eight vertices. Note that the corners for both surface and volume mapping can be placed on curves rather than vertices; this allows mapping surfaces and volumes with less than four and eight vertices, respectively. For example, the mapped quarter cylinder shown in Figure 2 has only five surfaces.

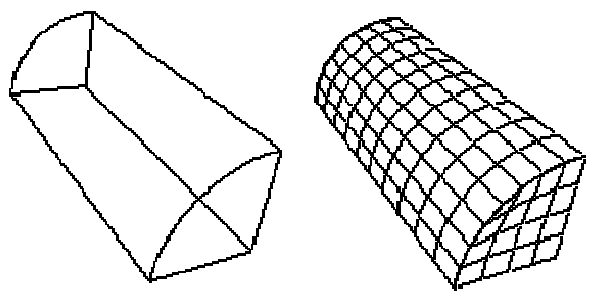


Figure 2. Volume Mapping of a 5-surfaced volume

Pave

Applies to: Surfaces

Summary: Automatically meshes a surface with an unstructured quadrilateral mesh.

Syntax:

Surface <range> Scheme Pave Related Commands:

[set] Paver Diagonal Scale <factor (Default = 0.9)> [set] Paver Grid Cell <factor (Default = 2.5)>[set] Paver LinearSizing {Off | ON} Surface <range> Sizing Function Type ...

[set] Paver Smooth Method {DEFAULT | Smooth Scheme | Old}

Discussion:

Paving (Blacker, 91; White, 97) allows the meshing of an arbitrary three-dimensional surface with quadrilateral elements. The paver supports interior holes, arbitrary boundaries, hard lines, and zero-width cracks. It also allows for easy transitions between dissimilar sizes of elements and element size variations based on sizing functions. Figure 1 shows the same surface meshed with mapping (left) and paving (right) schemes using the same discretization of the boundary curves.

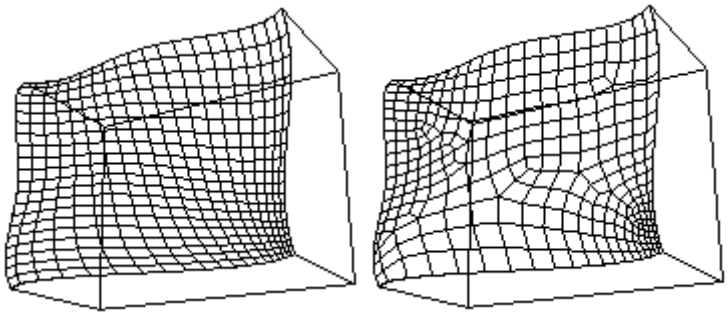


Figure 1. Map (left) and Paved (right) Surface Meshes

Element Shape Improvement

When meshing a surface geometry with paving, clean-up and smoothing techniques are automatically applied to the paved mesh. These methods improve the regularity and quality of the surface mesh. By default the paver uses its own smoothing methods that are not directly-callable from CUBIT. Using one of CUBIT's callable smoothing methods in place of the default method will sometimes improve mesh quality, depending on the surface geometry and specific mesh characteristics. If the paver produces poor element quality, switching the smoothing scheme may help. This is done by the command:

[set] Paver Smooth Method {DEFAULT | Smooth Scheme | Old}

When the "Smooth Scheme" is selected, the smoothing scheme specified for the surface will be used in place of the paver's smoother. See "Mesh Smoothing"for more information about the available smoothing schemes in CUBIT.

Controlling Flattening of Elements

The smoothers flatten elements, such as inserted wedges, that have two edges on the active mesh front. In meshes where this "corner" is a real corner, flattening the element may give an unacceptable mesh. The following command controls how much the diagonal of such an element is able to shrink.

[set] Paver Diagonal Scale <factor (Default = 0.9)>

The range of for the scale factor is 0.5 to 1.0. A scale factor of 1.0 will force the element to be a parallelogram as long as it is on the mesh front. A value of 0.5 will allow the diagonal to be half its calculated length. The element may became triangular in shape with the two sides on the mesh front being collinear.

Controlling the Grid Search for Intersection Checking

The paver divides the bounding box of a surface into a number of cells based on the average length of an element. It uses these cells to speed intersection checking of new element edges with the existing mesh. If both very long and very short edges fall in the same area, it is possible that a long edge which spans the search region is excluded from the intersection check when it does intersect the new element. The following command allows the user to adjust the size of the grid cells.

[set] Paver Grid Cell <factor (Default = 2.5)>

The grid cell factor is a multiplier applied to the average element size, which then becomes the grid cell size. The surface's bounding box is divided by this cell size to determine the number of cells in each direction. A larger cell size means each cell contains more nodes and edges. A smaller cell size means each cell has fewer nodes and edges. A larger cell size forces the intersection algorithm to check more potential intersections, which results in long paver times. A smaller cell size gives the intersection algorithm few edges to check (faster execution) but may result in missed intersections where the ratio of long to short element edges is great. Increase this value if the paver is missing intersections of elements.

Controlling the Paver Sizing Function

The paving algorithm will automatically select a "linear" sizing function if the ratio the largest element to the smallest is greater than 6.0 and no other sizing function is specified for the surface. This is usually desirable. When it is not, the user can change this behavior with the command:

[set] Paver LinearSizing {Off | ON}

Setting paver linear sizing to "off" will keep the default behavior. The size of the element will be based on the side(s) of the element on the mesh front. For a discussion of sizing functions, including how to automatically set up size transitions, see Adaptive Meshing.

Surface Vertex Types

- Surface Vertex Commands
- Listing and Drawing Vertex Types
- Triangle Vertex Types
- Adjusting the Automatic Vertex Type Selection Algorithm
- Volume Curve Types

Several meshing algorithms in CUBIT "classify" the vertices of a surface or volume to produce a high quality mesh. This classification is based on the angle between the edges meeting at the vertex, and helps determine where to place the corners of the map, submap or trimesh, or the triangles in the trimap or tripave schemes. For example, a surface mapping algorithm must identify the four vertices of the surface that best represent the surface as a rectangle. Figure 1 illustrates the vertex angle types for mapped and submapped surfaces, and the correspondence between vertex types and the placement of corners in a mapped or submapped mesh.

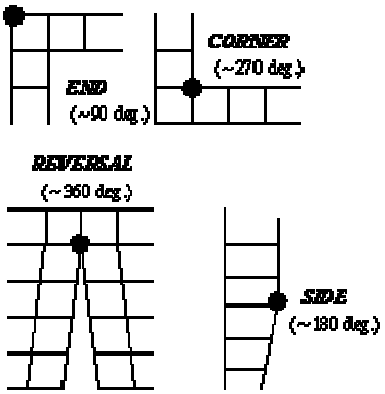


Figure 1. Angle Types for Mapped and Submapped Surfaces: An End vertex is contained in one element, a Side vertex two, a Corner three, and a Reversal four.

The surface vertex type is computed automatically during meshing, but can also be specified manually. In some cases, choosing vertex types manually results in a better quality mesh or a mesh that is preferable to the user. Vertex types have a firmness, just as meshing schemes do. Automatically selected vertex types are **soft**, while user-set vertex types are **hard**. Instead of a type, an angle in degrees can be specified instead.

Surface Vertex Commands

Vertex types are set using the following commands:

Surface <surface_id> Set [Vertex <vertex_id_range> [Loop_index <int>]] Type {End|Side|Corner|Reversal}

Surface <surface_id> Set Vertex [<vertex_id_range> [Loop_index <int>]] Angle <value>

Surface <surface_id> Set [Vertex <vertex_id_range> [Loop_index <int>]] Type {Default|Soft|Hard}

If no vertices are specified, the command is applied to all vertices of each surface. The **loop_index** is used only for vertices that are on the boundary of a single surface more than once.

Note that a vertex may be connected to several surfaces and its classification can be different for each of those surfaces.

The influence of vertex types when mapping or submapping a surface is illustrated in Figure 2. There, the same surface is submapped in two different ways by adjusting the vertex types of ten vertices.

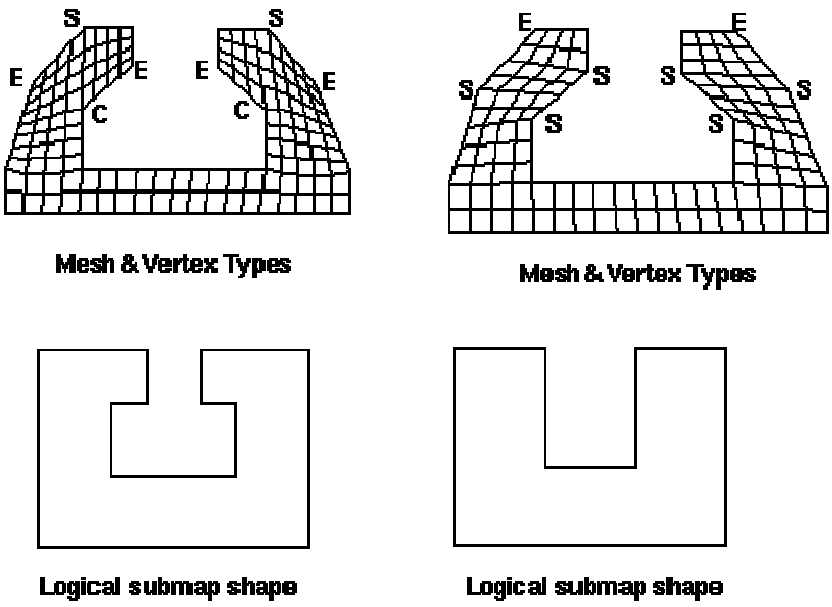


Figure 2. Influence of vertex types on submap meshes; vertices whose types are changed are indicated above, along with the mesh produced; logical submap shape shown below.

Listing and Drawing Vertex Types

Listing a surface lists the types of the vertices. The vertex type settings may also be drawn with the following commands:

Draw Surface <surface_id_range> {Vertex Angle|Vertex Type}

Triangle Vertex Types

For a surface that will be meshed with scheme trimap or tripave, the user may specify the angle below which triangles are inserted:

Surface <surface_id_range> Angle <angle>

The user may also set whether to add a triangle at a particular vertex:

Surface <surface_id> Set [Vertex <vertex_id_range> [Loop_index <int>]] Type {Triangle|Nontriangle}

Adjusting the Automatic Vertex Type Selection Algorithm

The user may specify the maximum allowable angle at a corner with the following command:

Set {Corner|End} Angle <degrees>

The user may also give greater priority to one automatic selection criteria over the others by changing the following absolute weights. The **corner weight** considers how large angles are at corners. The **turn weight** considers how L-shaped the surface is. The **interval weight** considers how much intervals must change. The **large angle weight** affects only auto-scheme selection: surfaces with a large angle will be paved instead. Each weight's default is 1 and must be between 0 and 10. The bigger a weight the more that criteria is considered.

Set Corner Weight <value>

Set Turn Weight <value>

Set Interval Weight <value>

Set Large Angle Weight <value>

An illustration of a mesh produced by the submapping algorithm is shown in Figure 2. The meshes produced by submapping on the left and right result from adjusting the vertex types of the eight vertices shown.

Volume Curve Types

When sweeping, a 2.5 dimensional meshing scheme, curves perpendicular to the sweep direction can have a type with respect to the volume. These types are usually automatically selected. The following commands are useful:

Draw Volume <surface_id_range> {Curve Angle|Curve Type}

List Volume <volume_id> Curve Type

Volume <volume_id> Set [Curve <curve_id_range>] Type {End|Side|Corner|Reversal}

Volume <volume_id> Set [Curve <curve_id_range>] Type {Default|Soft|Hard}

Pentagon

Applies to: Surfaces

Summary: Produces a pentagon-primitive mesh for a surface

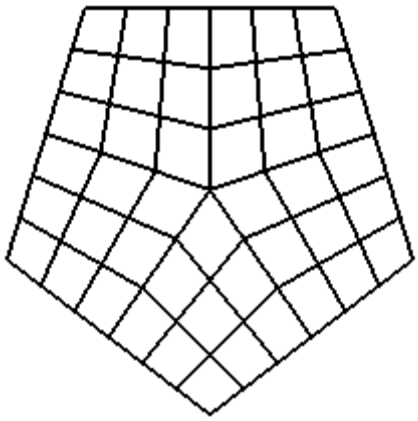
Syntax:

Surface <range> Scheme Pentagon

Discussion:

The pentagon scheme is a meshing primitive for 5-sided regions. It is similar to the triprimitive and polyhedron schemes, but is hard-coded for 5 sided surfaces.

The pentagon scheme indicates the region should be meshed as a pentagon. The scheme works best if the shape has 5 well-defined corners; however shapes with more corners can be meshed. The algorithm requires that there be at least 10 intervals (2 per side) specified on the curves representing the perimeter of the surface. In addition, the sum of the intervals on any three connected sides must be at least two greater than the sum of the intervals on the remaining two sides. Figure 1 shows two examples of pentagon meshes.



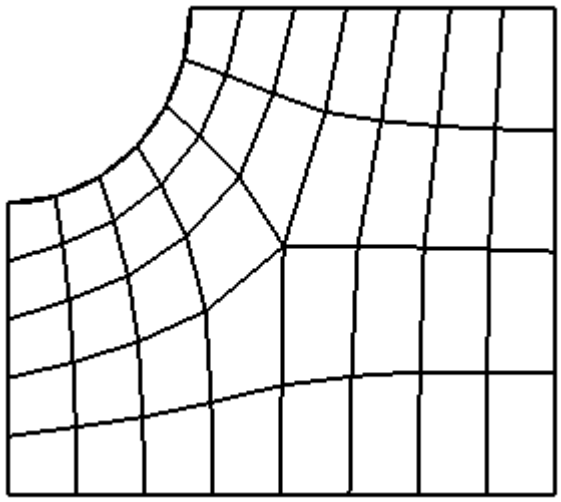


Figure 1. Examples of Pentagon Scheme Meshes

Pinpoint

Applies to: Curves

Summary:Meshes a curve with node spacing specified by the user.

Syntax:

Curve <range> Scheme Pinpoint Location <list of doubles>

Discussion:

The **Pinpoint** scheme allow the user to specify exactly where on a curve to place nodes. The list of doubles are absolute positions, measured from the start vertex. The user can enter as many as needed, and they do not need to be in numerical order. Below is an example of a curve that has been meshed using the following scheme:

curve 2 scheme pinpoint location 1 4 5 6 6.2 6.4 6.6 9:

Polyhedron

Applies to: Surfaces and Volumes.

Summary: Produces an arbitrary-sided block primitive mesh for a surface or volume.

Syntax:

Volume <range> Scheme Polyhedron

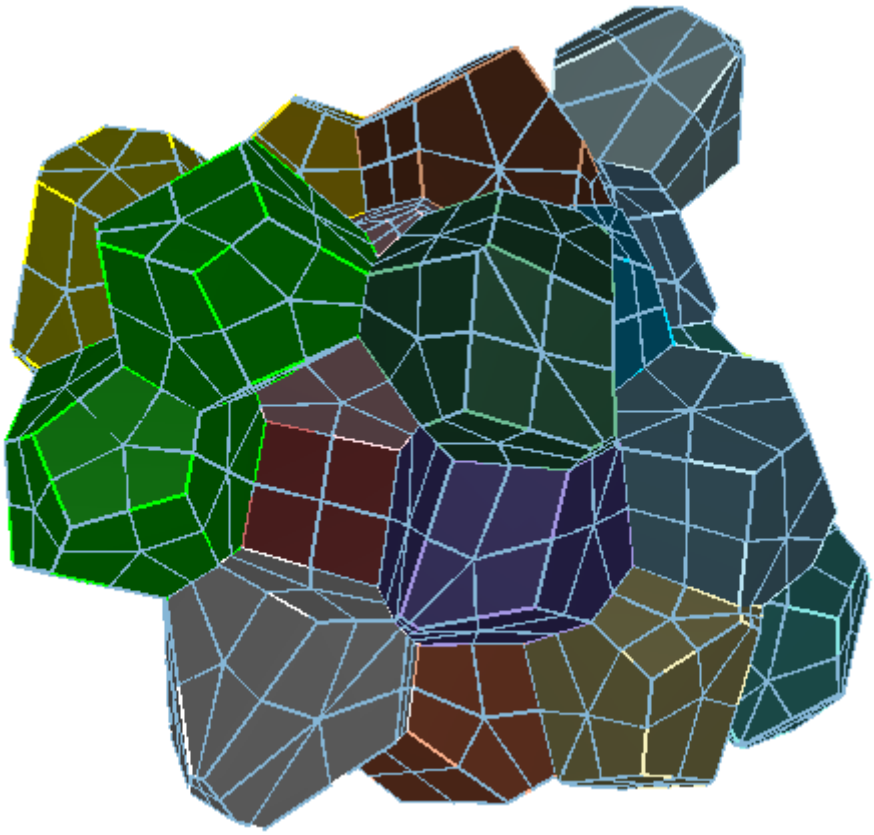
Surface <range> Scheme Polyhedron

Discussion:

The polyhedron scheme is a meshing primitive for 2d and 3d n-sided regions. This is similar to the triprimitive , tetprimitive, and pentagon schemes, except rather than 3, 4, or 5 sides, it allows an arbitrary number of sides. The scheme works best on convex regions. Surfaces must have only one loop, and each vertex must be connected to exactly two curves on the surface (e.g., no hardlines). Volumes must have only one shell, each vertex must be connected to exactly three surfaces on the volume, and each surface should be meshed with scheme polyhedron. There are some interval assignment requirements as well, which should be automatically handled by CUBIT.

If the polyhedron scheme is specified for the volume, then the surfaces of the volume are automatically assigned scheme polyhedron as well, unless they were hard-set by the user. Schemes should be specified on all volumes of an assembly prior to meshing any of them. Scheme polyhedron attaches extra data to volumes; if Cubit is behaving strangely, the user may need to explicitly remove that data with a **reset volume all**, or similar command.

Scheme polyhedron was designed for assemblies of material grains, where each volume is roughly a Voronoi region, and the assembly is a periodic space-filling model (tile). Figure 1 shows two examples of polyhedron meshes.



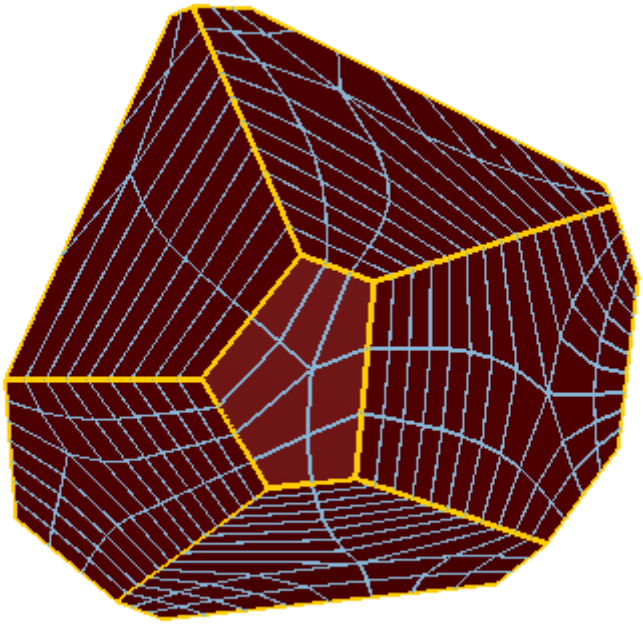


Figure 1. Examples of Polyhedron Scheme Meshes

Sphere

Applies to: Volumes topologically equivalent to a sphere and having one surface.

Summary: Generates a radially-graded hex mesh on a spherical volume.

Syntax:

> Volume <range> Scheme Sphere [Graded_interval <int>] [Az_interval <int>] [Bias <val>] [Fraction <val>]

Discussion:

This scheme generates a radially-graded mesh on a spherical volume having a single bounding surface. The mesh is a straightforward generalization of the circle scheme for surfaces. The number of azimuthal intervals around the equator is controlled by the az_interval input parameter. The number of radial intervals in the outer portion of the sphere is controlled by the graded_interval input parameter. Azimuthal mesh lines in the outer portion of the sphere have constant radius. The inner portion of the volume mesh forms a cube. The bias parameter controls the amount of radial grading in the outer portion of the mesh (default=1 gives a uniform mesh). The fraction parameter (between 0 and 1) determines what fraction of the sphere is occupied by the inner cube.

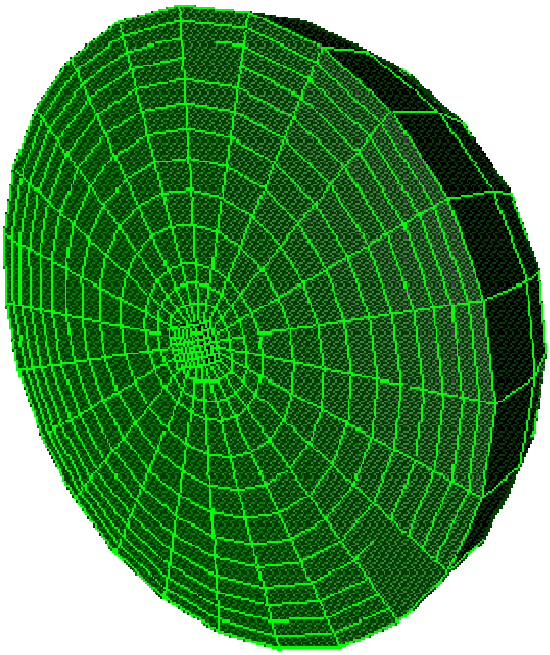


Figure 1. Sphere Scheme Example

STransition

Applies to: Surfaces

Summary:

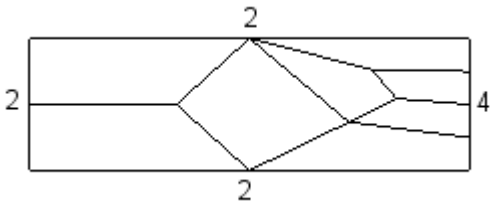
Produces a simple transitional mapped mesh.

Syntax:

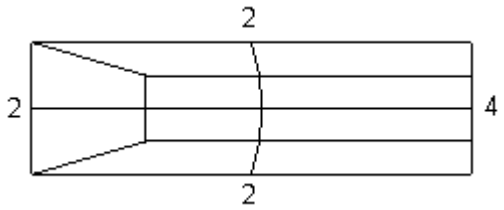
Surface <surface_id_range> Scheme STransition

Discussion:

The STransition scheme transitions a mesh from one element density to another across a surface. This scheme is particularly helpful when the Paving scheme produces a poor mesh. The following two figures show a specific case where the STransition scheme may offer an improvement.

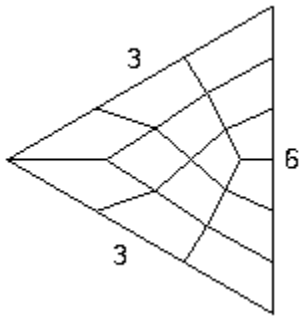


Pave scheme



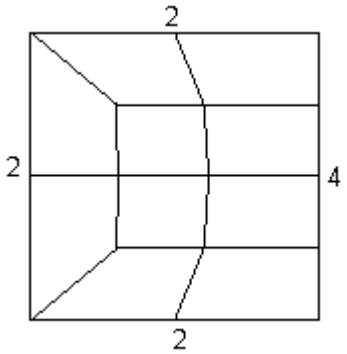
STransition scheme

For triangular surfaces, the STransition scheme will produce similar results when compared to the Triprimitive scheme. However, STransition is capable of handling more varied interval settings. The following triangle fails when using the Triprimitive scheme but succeeds with the STransition scheme.

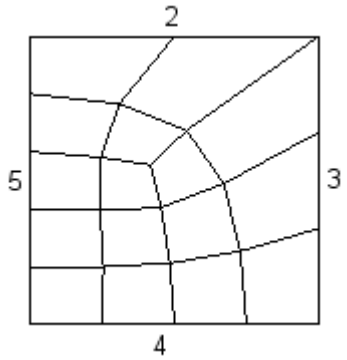


STransition scheme on a triangular surface with intervals set to 3, 3, and 6.

The figures below show the STransition meshing scheme response to different shapes and interval settings.

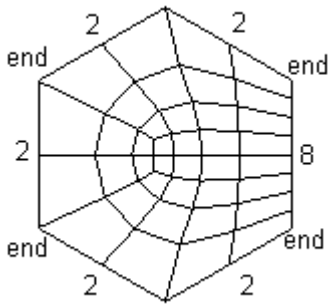


STransition scheme on a rectangular surface with three intervals set to 2 and one set to 4.



STransition scheme on a rectangular surface with intervals set to 2, 3, 4, and 5.

The user also has the option of specifying END or SIDE surface vertex types.



STransition scheme on a hexagon surface with five intervals set to 2, one interval set to 8, and user specified endpoints.

Note, that the Centroid Area Pull smoothing algorithm sometimes gives better results than the default Winslow smoothing algorithm for STransition meshes.

Submap

Applies to: Surfaces, Volumes

Summary: Produces a structured mesh for surfaces/volumes with more than 4/6 logical sides

Syntax:

{Surface|Volume} <range> Scheme Submap

Related Commands:

{Surface|Volume} <range> Submap Smooth <on|off>

Discussion:

Submapping (Whiteley, 96) is a meshing tool based on the surface mapping capability discussed previously, and is suited for mesh generation on surfaces which can be decomposed into mappable subsurfaces. This algorithm uses a decomposition method to break the surface into simple mappable regions. Submapping is not limited by the number of logical sides in the geometry or by the number of edges. The submap tool, however is best suited for surfaces and volumes that are fairly blocky or that contain interior angles that are close to multiples of 90 degrees.

An example of a volume and its surfaces meshed with submapping is shown in Figure 1.

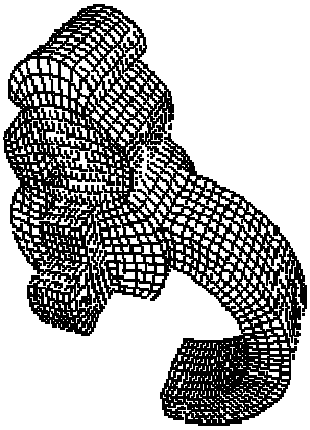


Figure 1. Quadrilateral and Hexahedral meshes generated by submapping

Like the mapping scheme, submapping uses vertex types to determine where to put the corners of the mapped mesh (See Surface Vertex Types). For surface submapping, curves on the surface are traversed and grouped into " logical sides " by a classification of the curves position in a local "i-j" coordinate system.

Volume submapping uses the logical sides for the bounding surfaces and the vertex types to construct a logical "i-j-k" coordinate system, which is used to construct the logical sides of the volume. For surface and volume submapping, the sides are used to formulate the interval constraints for the surface or volume.

Figure 2 shows an example of this logical classification technique, where the edges on the front surface have been classified in the i-j coordinate system; the figure also shows the submapped mesh for that volume.

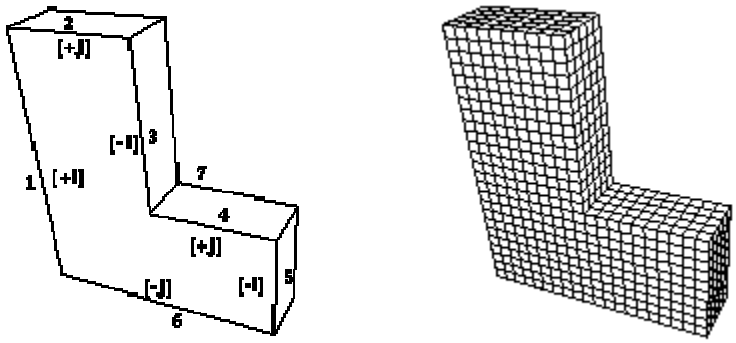


Figure 2. Scheme Submap Logical Properties

After submapping has subdivided the surface and applied the mapped meshing technique mentioned above, the mesh is smoothed to improve mesh quality. Because the decomposition performed by submapping is mesh based, no geometry is created in the process and the resulting interior mesh can be smoothed. Sometimes smoothing can decrease the quality of the mesh; in this case the following command can turn off the automatic smoothing before meshing:

{Surface|Volume} <range> Submap Smooth <on|off>

Surface submapping also has the ability to mesh periodic surfaces such as cylinders. An example of a periodic surface meshed with submapping is shown in Figure 3. The requirement for meshing these surfaces is that the top and bottom of the cylinder must have matching intervals.

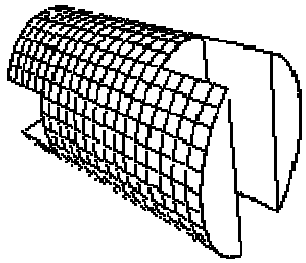


Figure 3. Periodic Surface Meshing with Submapping

For periodic surfaces, there are no curves connecting the top and bottom of the cylinder. Setting intervals in this direction on the surface can be done by setting the periodic interval for that surface (see Interval Assignment. No special commands need to be given to submap a periodic surface, the algorithm will automatically detect the fact that the surface is periodic. Currently, periodic surfaces with interior holes are not supported.

Stretch

Applies to: Curves

Summary: Permits user to specify the exact size of the first and/or last edges on a curve.

Syntax:

Curve <range> Scheme Stretch [First_size <double>] [Start Vertex <id>]

Curve <range> Scheme Stretch [First_size <double>] [Last_size <double>] [Start Vertex <id>]

Curve <range> Scheme Stretch [Stretch_factor <double>] [Start Vertex <id>]

Related Commands:

Scheme Bias and Dualbias.

Discussion:

This scheme allows the user to specify the exact length of the first and/or last edge on a curve mesh. Intermediate edge lengths will vary smoothly between these input values. Reasonable values for these parameters should be used (for example, the sizes must be less than the total length of the curve). If last_size is input, first_size must be input also. If stretch_factor is input, neither first_size nor last_size can be input. This scheme does not currently work on periodic curves.

Sweep

Applies to: Volumes

Summary: Produces an extruded hexahedral mesh for 2.5D volumes.

Syntax:

Volume <range> Scheme Sweep [Source [Surface] <range>] [Target [Surface] <range>][Rotate {on | OFF}]Volume <range> Scheme Sweep Vector <xval yval zval>

Related Commands:

Volume <range> Sweep Smooth [AUTO|copy|linear|residual|winslow][set] Multisweep Smoothing {ON|off}Multisweep Volume <range> Remove

Discussion:

The sweep algorithm (Knupp, 98) can sweep general 2.5D geometries and can also do pure translation or rotations. A 2.5D geometry is characterized by source and target surfaces which are topologically similar. The hexahedral mesh is swept (extruded) between source and target along a single logical axis. Bounding the swept hexahedra between source and target surfaces, are the linking surfaces. Figures 1 and 2 show examples of source, target and linking surfaces.

Command Options: The user can specify the source and target surfaces. The user can also specify a geometric vector approximating the sweep direction, and let CUBIT determine the source and target surfaces. The user can specify just the source surfaces, and let cubit guess the target, or "scheme auto" can also be used.

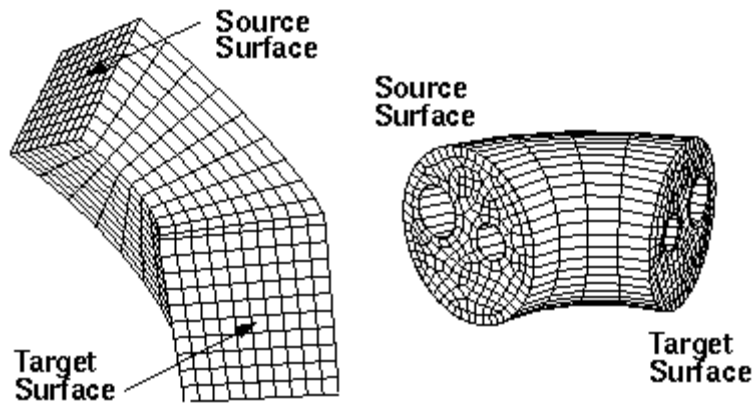


Figure 1. Sweep Volume Meshing

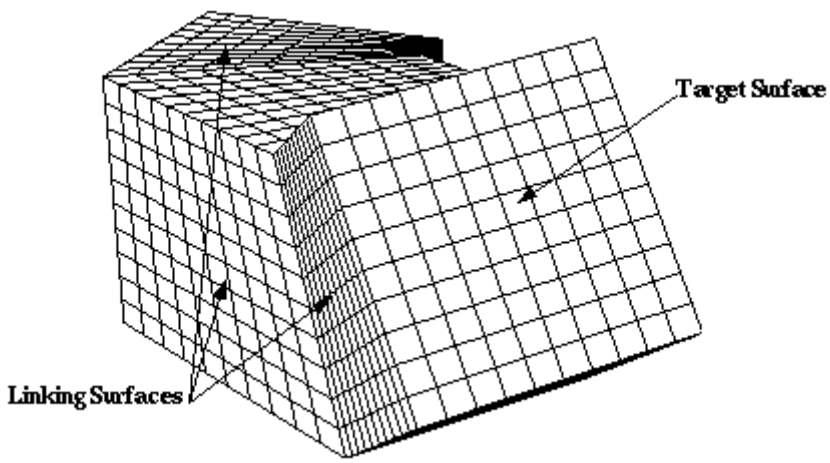


Figure 2. Multiple Linking Surface Volume Meshing with Scheme Sweep

In general, the procedure for using the sweep scheme is to first mesh the source surfaces. Any surface meshing scheme may be employed. Figure 1 displays swept meshes involving mapped and paved source surfaces. Linking surfaces must have either mapping or submapping schemes applied. The sweep algorithm can also handle multiple surfaces linking the source surface and the target surfaces. An example of this is shown in Figure 2. Note that for the multiple- linking-surface meshing case, the interval requirement is that the total number of intervals along each multiple edge path from the source surface to the target surface must be the same for each path. Once the appropriate mesh is applied to the source surface and intervals assigned, the **mesh** command may be issued.

In many cases auto-scheme selection can simplify this process by recognizing sweepable geometries and automatically select source and target surfaces. If the source and target surfaces are not specified, CUBIT attempts to automatically select them. CUBIT also automatically sets curve and vertex types in an attempt to make the mesh of the linking surfaces lead from a source surface to a target surface. These automatic selections may occasionally fail, in which case the user must manually select the source/target surfaces, or some of the curve and vertex types. After making some of these changes, the user should again set the volume scheme to sweep and attempt to mesh.

Occasionally the user must also adjust intervals along curves, in addition to the usual surface interval matching requirements. For a given pair of source/target surfaces, there must be the same number of hexahedral layers between them regardless of the path taken. This constrains the number of edges along curves of linking surfaces. For example, in Figure 1 right, the number of intervals through the holes must be the same as along the outer shell.

Rotate Option: The rotate option of sweeping is a specialized surface meshing option to map polar grids on curved linking surfaces. The rotate option is most effective when sweeping produces undesirable results due to element biasing on linking surfaces. The rotate option requires that linking surfaces be enclosed by four curves and that surfaces are not meshed prior to sweeping. The scheme creates node pairs on opposite linking curves and places interior surface nodes linearly between the pairs. Node spacing between the pairs is a proportion of the node spacing found on the source curve of the linking surface. **Figure 3** provides an example where sweeping was unable to produce a suitable mesh for a curved surface, but using the rotate option a polar grid is created for the linking surface.

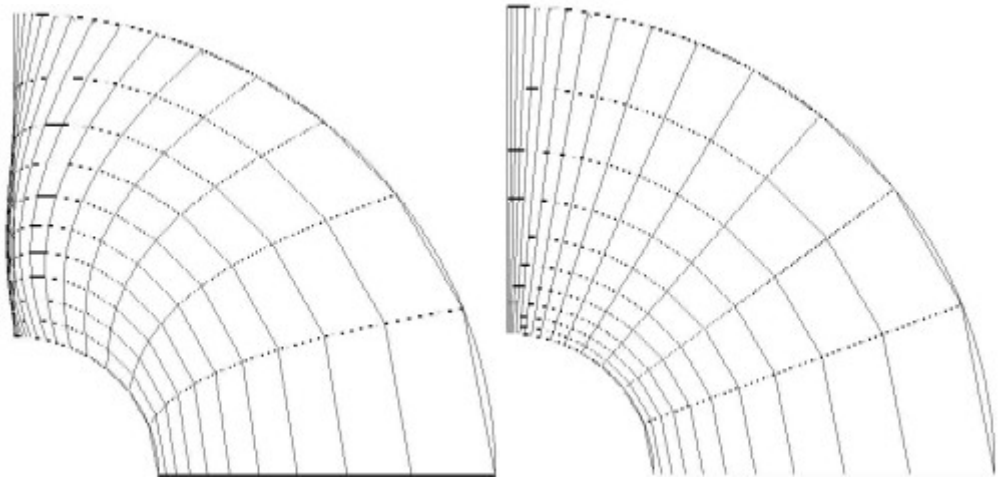
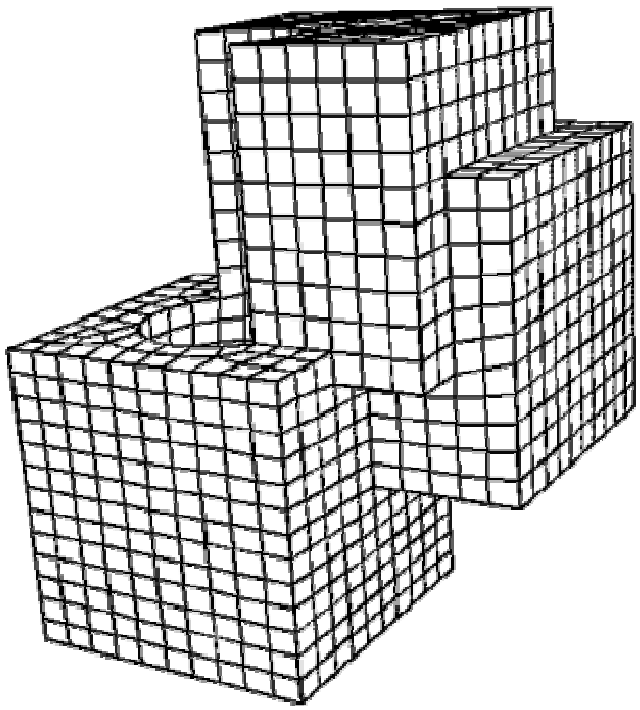


Figure 3. Example of where the sweep rotate option would be best suited. Figure on left shows mesh without the rotate option used.

Multisweep

While the basic sweeping algorithm requires only a single source and single target surface. The sweeping algorithm can also handle multiple source and target surfaces. The multisweep algorithm works by recognizing possible mesh and topology conflicts between the source and target surfaces and works to resolve these conflicts through the use of the virtual geometry capabilities in CUBIT. Figure 4 shows some examples of volumes which have been meshed with the multisweep algorithm.



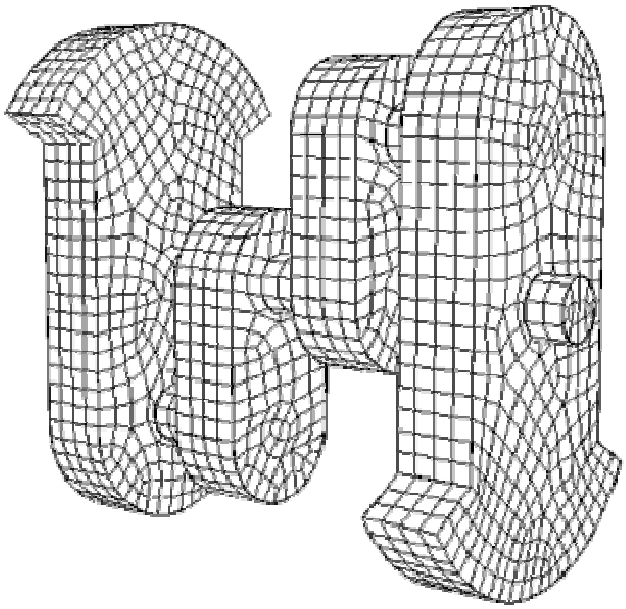


Figure 4. Examples of Multi-sweep meshes.

The multisweep algorithm is an addition to the regular sweeping algorithms, and is accessed by specifying scheme sweep and assigning multiple surfaces in the target surface list. In addition, the autoscheme selection algorithm may, also, assign some volumes to be multiswept.

As part of the multisweep process, CUBIT may automatically generate virtual geometric entities (curve or surface partitions). These virtual definitions will remain after multisweep is completed. The new virtual entities can be used on adjacent volumes for decomposing and aligning the mesh. Changes made to the geometry during multisweep can be removed (and the mesh left in place) with the "multisweep remove" command, the virtual geometry will not be saved with it. To remove the newly created virtual geometry, the following command may be used:

Multisweep Volume <range> Remove

Because the multisweep algorithm may alter some of the surface geometry on the volume, it is generally a good idea to attempt to mesh the multisweep volumes first before meshing any other volumes. Also note that this virtual geometry modification may also require some additional scheme selection and interval matching on adjoining volumes.

Smoothing Swept Meshes

Swept meshes are created layer by layer, the layers consisting of quadrilateral elements. To ensure adequate mesh quality, it may be necessary to smooth the nodes within each layer. The sweep tool permits five types of layer smoothing which can be selected by issuing the following command prior to meshing a volume whose scheme has been set to sweep:

Volume <range> Sweep Smooth [AUTO|copy|linear|residual|winslow]

Linear: If this option is selected, no layer smoothing is performed. The node positions are determined strictly by the affine transformation from the previous layer. Good quality swept meshes can be constructed using "linear" provided the volume geometry and meshed linking surfaces permit the volume mesh to be created by a translation, scaling, and/or rotation of the source mesh. Volumes for which this is nearly true may also produce acceptable quality with "linear". As one would expect, this option generates swept meshes more quickly than the other sweep smooth options. For compatibility with earlier versions of CUBIT, the keyword “off” can be used in place of the keyword “linear” for this command.

Winslow: If the "linear" option does not generate sufficient mesh quality, the "winslow" option is often successful. The winslow option smooths each layer using an elliptic smoother designed for unstructured meshes. While it is considerably slower than the "linear" option, much better mesh quality often results. Use this option for example, when the sweep cross section is variable in shape.

Copy: Smooth scheme "copy" smooths each layer using a weighted version of the Winslow smoother. The weights are computed from the source mesh; they help maintain any biased spacing that occurs on the source mesh. An example of when one might want to use the "copy" option would be a biased source mesh that was created using scheme circle. The biasing of the outer elements of the source mesh will be destroyed in the next sweep layer if scheme "winslow" is used, whereas it will be preserved with scheme "copy". For compatibility with earlier versions of CUBIT, the keyword “on” can be used in place of the keyword “copy” for this command.

Residual: The “residual” method is often used for meshing volumes that cannot be swept using the “linear” method. It tends to produce better quality meshes than the “linear” method while running faster than the Winslow-based smoothers (the “copy” or “winslow” methods). The sweeping algorithm uses an affine transformation to calculate the interior nodes’ positions of each new layer, but the mesh on the linking surface determines the positions of the nodes on the boundary of the layer. For the “residual” method, CUBIT calculates corrective adjustments for interior nodes using the “residuals” from boundary nodes. The “residual” is defined as the distance between the boundary node’s position (as determined by the surface mesh) and the boundary node’s ideal position (as determined by the affine transformation of the previous layer).

AUTO: This is the default option for the sweep smooth command. Smooth scheme "auto" causes the Sweep Tool to automatically choose between "linear", “residual”, and "winslow", layer by layer. Auto will choose "linear" if the layer needs no smoothing, “residual” if the layer needs a small amount of smoothing, and "winslow" otherwise. For volumes that have some layers which need smoothing and some that don't, "auto" will run faster than "winslow" because it won't smooth every layer. An example would be a volume that can at first be swept by a simple translation of the source mesh, but later transitions to a variable cross-section. Although "auto" will run faster than "winslow", it may result in quality that is inferior to "winslow". On the other hand, "auto" may run slower than "linear", but result in better quality. Scheme "auto" does not guarantee that no negative Jacobians are produced.

The "sweep smooth" command cannot be used except in conjunction with mesh sweeping.

If none of these smooth schemes result in adequate mesh quality, one can consider trying one of the volume smoothing schemes such as condition_number

Users who do not wish to experiment with these five options until they obtain adequate mesh quality are also encouraged to consider the autosmooth options.

Smoothing on volumes that use the multisweep algorithm can be controlled by the following command:

[set] Multisweep Smoothing {ON|off}

Some helpful hints in using sweep

1. Sweep runs faster if "sweep smooth" is off. If the geometry/surface mesh permits translation, rotation, or scaling then no smoothing should be needed.
2. The source and linking surfaces of the volume will be automatically meshed if the user has not already meshed them prior to meshing the volume with sweep. It is important to have high quality meshes on the linking surfaces that are synchronized with one another to that sweep can succeed. For example, if the geometry suggests translation as the appropriate technique, a translated mesh will still not result from sweep unless the meshes on the volume surfaces are set up accordingly. If there are bad quadrilaterals on the surface meshes, sweep automatically aborts.
3. The target may be meshed by the user or that task may be left to sweep. If the target surface is meshed prior to invoking sweep, then the target mesh must be topologically equivalent to the set of source surface meshes.
4. Biasing of the curve meshes in the direction of the sweep is preserved by the sweep. Biasing of the source mesh boundary is not preserved under a sweep. To accomplish the latter, the user must bias the target surface boundary.
5. The most common error message generated by sweep reads "Target partially reached. Check intervals on Linking Surfaces." The error-trap that provokes this message is quite general and may occur for a number of reasons, not necessarily the reason given. One of the most frequent causes for this message is a geometry with a thru-hole with the linking surfaces having a different number of intervals on the inside vs. the outside of the volume.
6. If either or both the source and/or target surfaces are omitted from the scheme setting command, CUBIT will determine source and target surfaces (See Automatic Scheme Selection). Sweeping can be further automated using the "sweep groups" command.
7. Limitations: Not all geometries are sweepable. Even some that appear sweepable may not be, depending on the linking surface meshes. Highly curved source and target surfaces may not be meshable with the current sweep algorithm.

Autosmooth

When creating large meshes, or doing meshing of assemblies, often a greater amount of automation is desired. With this object in mind, the autosmoothing command was added to perform the same meshing process that is typically done by a user on each volume of an assembly. The steps for completing a mesh on an assembly of volumes typically follows the rough outline:

1. Generate the swept mesh without using any sweep smooth options. Check the quality of the resulting mesh. If the quality is poor, delete the mesh and proceed to step 2.
2. Generate the swept mesh using the sweep smooth option winslow. Check the quality of the resulting mesh. If the quality is poor, continue with step 3.
3. Smooth the mesh on the target surfaces, then use the condition number smoother to improve the quality of the volume elements. Check the resulting quality.

The autosmooth command is an attempt to automate this process to reduce the amount of user interaction required during meshing. When autosmooth is turned on, the outline above is followed until a reasonable quality mesh is produced. If step 3 is completed above without producing a quality mesh, then the user is required to further decompose the model, or choose a different meshing scheme.

The following is the command syntax for activating autosmoothing:

Volume {Default|<range>} Autosmooth {OFF|on}

Volume <range> Autosmooth Target {OFF|on}

The default option for this command is set to off, simply to decrease the potential amount of time that the user might experience when performing test meshes. Setting the volume default option in the **.cubit** initialization file will force all sweeping operations in CUBIT to go through the steps outlined above. Optionally, you can enable this for specific volumes only.

Grouping Sweepable Volumes

Swept meshing relies on the constraint that the source and target meshes are topologically identical or the target surface is unmeshed. This results in there being dependencies between swept volumes connected through non-manifold surfaces; these dependencies must be satisfied before the group of volumes can be meshed successfully. For example, if the model was a series of connected cylinders, the proper way to mesh the model would be to sweep each volume starting at the top (or bottom) and continuing through each successive connected volume.

With larger models and with models that contain volumes that require many source surfaces, the process of determining the correct sweeping ordering becomes tedious. The sweep grouping capability computes these dependencies and puts the volumes into groups, in an order which represents those dependencies. The volumes are meshed in the correct order when the resulting group is meshed.

To compute the sweep dependencies, use the command:

Group Sweep Volumes

This will create a group named "sweep_groups", which can then be meshed using the command:

Mesh sweep_groups

TetMesh

Applies to: Volumes

Summary: Automatically meshes a volume with an unstructured tetrahedral mesh.

Syntax:

Volume <range> Scheme {TetMesh|TetINRIA}

Related Commands:

THex Volume All

Volume <volume_id> Tetmesh Respect {Face|Tri|Edge|Node} <range>

Discussion:

The TetMesh scheme fills and arbitrary three-dimensional volume with tetrahedral elements. The surfaces are first triangulated with one of the triangle schemes (TriMesh or TriAdvance) or a quadrilateral scheme with the quadrilaterals being split into two triangles.

The Simulog/INRIA tet-mesher is included in CUBIT. This is a robust and fast tetrahedral mesher developed in France at INRIA and distributed by Simulog. Figure 1 shows a volume filled with tetrahedra by this algorithm. You can specify this scheme for a volume by giving either scheme TetMesh or TetINRIA, as these two scheme names are synonymous.

Using tets as the basis of an unstructured hexahedral mesh

Tet meshing can be used to generate hexahedral meshes using the THex command. Each of the tetrahedron can be converted into 4 hexes, producing a fully conformal hexahedral mesh, albeit of poorer quality. These meshes can often be used in codes that are less sensitive to mesh quality and mesh directionality. The THex command requires that all tets in the model be converted to hexahedra with the same command.

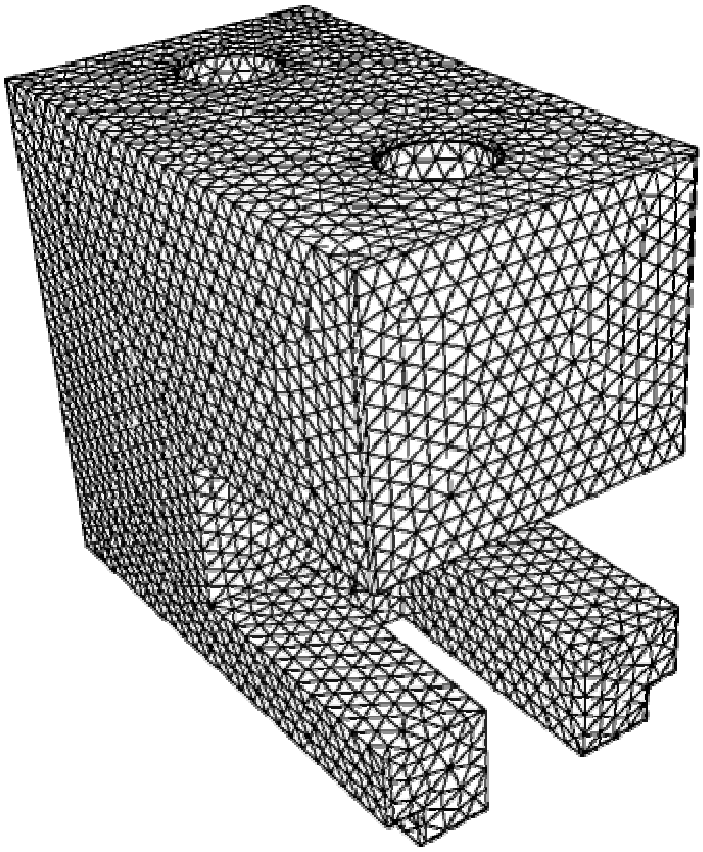


Figure 1. Tetrahedral Mesh generated with the TetInria scheme. Surface meshing was performed with the TriAdvance scheme.

Conforming the tetmesh to internal features

In some cases it is necessary for the finite element mesh to conform to internal features of the model. The tetmesh scheme provides this capability provided the tetmesh respect command has been previously issued to define the features that will be respected.

Volume <volume_id> Tetmesh Respect {Face|Tri|Edge|Node} <range>

The tetmesh respect command allows the user to specify mesh entities that will be part of a tetrahedral mesh. These faces, triangles, edges, or nodes are inside the volume since all surface mesh features will appear in the final tetrahedral mesh by default. These mesh entities specified to be respected can be generated from other meshing commands on free vertices, curves, or surfaces.

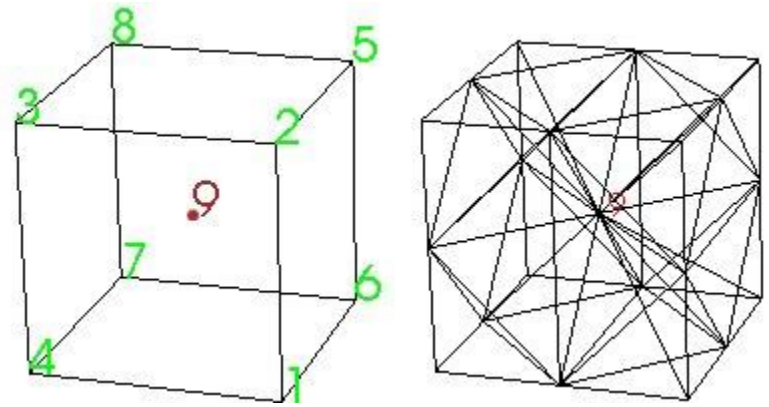


Figure 2. Example of using tetmesh respect to ensure node 9 is captured in the tetmesh.

For example, Figure 2 is an example of using the tetmesh respect command to enforce a node at the center of a cube. Node 9 in this example was generated by first creating a free vertex at the center location and meshing the vertex. (mesh vertex 9). The following commands would then be used to generate the tetmesh that respected node 9.

```
volume 1 scheme tetmesh
tetmesh respect node 9
mesh volume 1
```

The tetmesh respect command can also be used to enforce multiple mesh entities. To accomplish this, the tetmesh respect command may be issued multiple times. For example, If node12 and a triangle 2 inside volume 3 was to appear in the volumetric mesh, the following commands could be used:

```
volume 3 scheme tetmesh
volume 3 tetmesh respect node 12
volume 3 tetmesh respect tri 2
mesh volume 1
```

Tetprimitive

Applies to: Volumes

Summary: Meshes a 4 "sided" object with hexahedral elements using the standard tetrahedron primitive.

Syntax:

```
Volume <range> Scheme Tetprimitive [Combine Surface <range>] [Combine Surface <range>]
[Combine Surface <range>] [Combine Surface <range>]
```

Discussion:

The tetprimitive scheme is used to create a hexahedral mesh in a volume which fits the shape of a tetrahedral primitive. The **Tetprimitive** scheme assumes that each of the four surfaces have been meshed with the triprimitive, or similar, meshing scheme. If more than four surfaces form the tetrahedron geometry, the surfaces forming a logical side can be combined using the **combine** option.

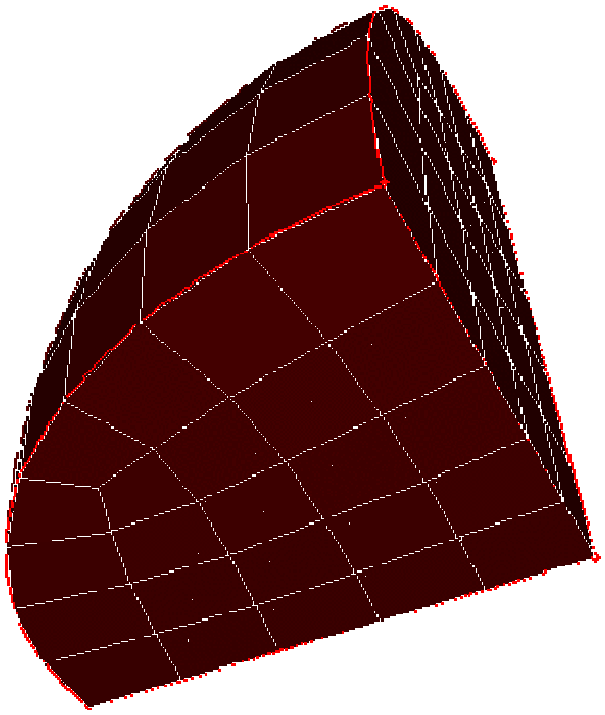


Figure 1. Sphere octant hex meshed with scheme Tetprimitive, surfaces meshed using scheme Triprimitive

TriDelaunay

Applies to: Surfaces

Summary: Automatically meshes planar surface geometry with triangle elements.

Syntax:

Surface <range> Scheme TriDelaunay

Discussion:

The scheme TriDelaunay is an alternative triangle meshing scheme to the TriAdvance and QTri schemes. This algorithm uses the Delaunay [Watson,81] criterion for connecting nodes into triangles. It also utilizes the Guaranteed Quality [Ruppert,92] approach for inserting nodes into the mesh. Because of the inherent nature of the Delaunay criterion, this scheme is limited only to planar surfaces.

TriDelaunay can also utilize a sizing function if one is defined for the surface.

Note that if an attempt is made to mesh a surface that is non-planar, a error will be generated. Use scheme TriMesh, TriAdvance or QTri to mesh non-planar surfaces.

TriMap

Applies to: Surfaces

Summary: Places triangle elements at some vertices, and map meshes the remaining surface.

Syntax:

Surface <range> Scheme Trimap

Related Commands:

Surface <range> Vertex <range> Type {triangle|notriangle}

Discussion:

Some surfaces contain bounding curves which meet at a very acute angle. Meshing these surfaces with an all-quadrilateral mesh will result in a very skewed quad to resolve that angle. In some cases, this is a worse result than simply placing a triangular element to resolve that angle. This scheme resolves this situation by placing a triangular element in these tight corners, and filling the remainder of the surface with a mapped mesh.

The algorithm can automatically compute whether a triangular element is necessary, along with where to place that element. To override the choice of where triangular elements are used, the following command can be issued:

Surface <range> Vertex <range> Type {triangle|notriangle}

TriMesh, TriAdvance

Applies to: Surfaces

Summary: Automatically meshes surface geometry with triangle elements.

Syntax:

Surface <range> Scheme {TriMesh|TriAdvance}

Discussion:

The triangle meshing schemes fill an arbitrary surface with triangle elements. Two algorithms are available for this purpose.

1. The scheme **TriAdvance** is an advancing front algorithm which allows holes in the surface and transitions between dissimilar element sizes. It can use a sizing function like the pave scheme if one is defined for the surface. Future development will add hard lines to this scheme's capabilities. You specify this scheme for a surface by giving the command:

Surface <range> Scheme TriAdvance

2. The scheme **TriMesh** automatically switches between the TriAdvance and the QTri schemes. First, it tries the **TriAdvance** scheme; if that fails, it tries the QTri scheme. The QTri scheme first paves the surface and then cuts the quadrilateral elements in half to form triangles. Figure 2 shows an example of a mesh created with the QTri method.

You specify this scheme for a surface by giving the command:

Surface <range> Scheme TriMesh

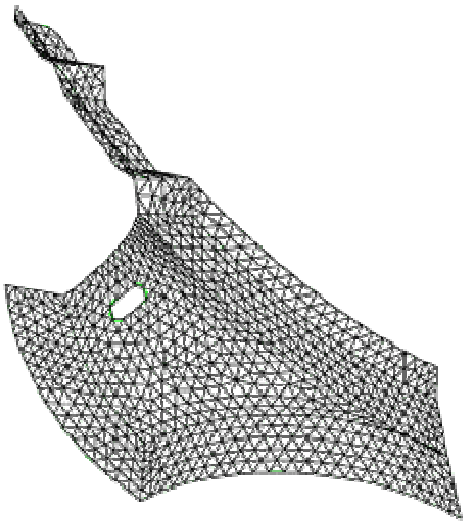


Figure 1. Triangle mesh generated with scheme TriAdvance

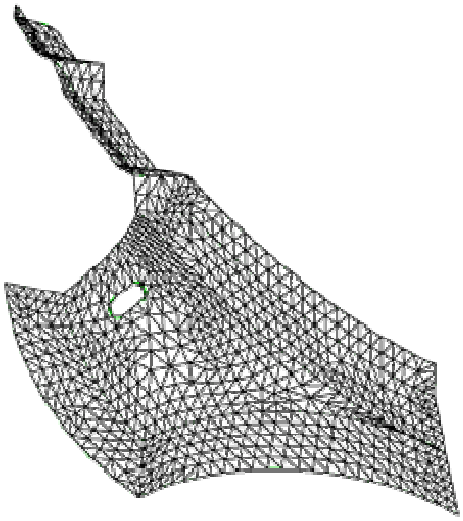


Figure 2. Triangle mesh generated with QTri scheme

TriPave

Applies to: Surface

Summary: Places triangle elements at some vertices, and paves the remaining surface.

Syntax:

Surface <range> Scheme Tripave

Related Commands:

Surface <range> Vertex <range> Type {triangle|notriangle}

Discussion:

Similar to the trimap algorithm, but uses paving instead of mapping to fill the remainder of the surface with quadrilaterals.

TriPrimitive

Applies to: Surfaces

Summary: Produces a triangle-primitive mesh for a surface with three logical sides

Syntax:

Surface <range> Scheme Triprimitive [SMOOTH | nosmoothing]

Discussion:

The triprimitive scheme indicates that the region should be meshed as a triangle. A surface may use the triprimitive scheme if three "natural", or obvious, corners of the surface can be identified. For instance, the surface of a sphere octant (shown in the figure below) is handled nicely by the triprimitive scheme. The algorithm requires that there be at least 6 intervals (2 per side) specified on the curves representing the perimeter of the surface and that the sum of the intervals on any two of the triangle's sides be at least two greater than the number of intervals on the remaining side. The following figure illustrates a triprimitive mesh on a 3D surface.

By default, the triprimitive algorithm will smooth the mesh with an iterative smoothing scheme. This smoothing can be disabled by using the "nosmoothing" option with this command. The quality of the mesh will often be significantly degraded by disabling smoothing, but in certain cases the unsmoothed mesh may be preferred.

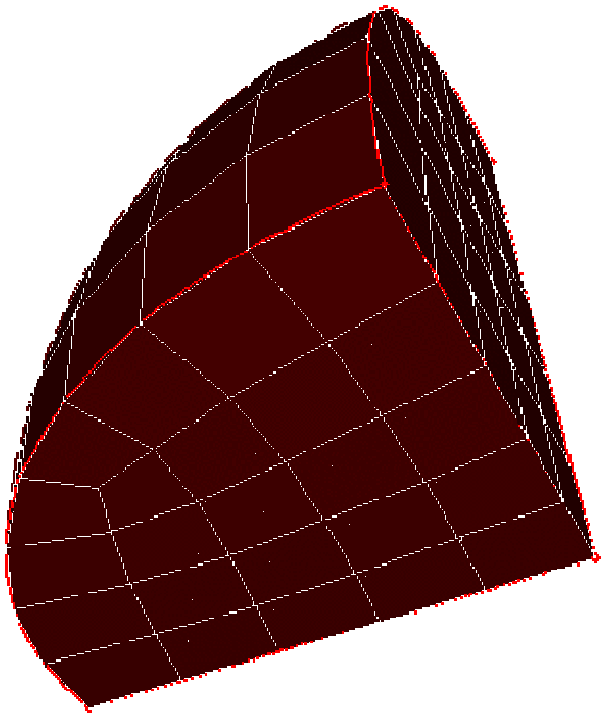


Figure 1. Surfaces meshed with scheme Triprimitive

Radialmesh

Summary: Creates a free cylindrical mesh with precise node locations based on input radii, angles, and offsets, then creates mesh-based geometry to fit the mesh.

Syntax:

```
Create Radialmesh \
   numZ <val> [span <val>] \
      zblock 1 [<offset val>] \
            {interval|bias|fraction|first size} <val> \
           [{interval|bias|fraction|last size} <val>] \
      zblock 2 [<offset val>] \
            {interval|bias|fraction|first size} <val> \
           [{interval|bias|fraction|last size} <val>] \
      ... numZ \

   numR <val> {trisection|initial radius<val>} \
      rblock 1 <offset radius val> \
            {interval|bias|fraction|first size} <val> \
           [{interval|bias|fraction|last size} <val>] \
      rblock 2 <offset radius val> \
            {interval|bias|fraction|first size} <val> \
           [{interval|bias|fraction|last size} <val>] \
      ... numR \

   numA <val> [full360] [span <val>] \
      ablock 1 [<offset angle val>] \
            {interval|bias|fraction|first angle} <val> \
           [{interval|bias|fraction|last angle} <val>] \
      ablock 2 [<offset angle val>] \
            {interval|bias|fraction|first angle} <val> \
           [{interval|bias|fraction|last angle} <val>] \
      ... numA
```

Discussion:

The purpose of the **radialmesh** command is to create a cylindrical mesh with precise node locations. Unlike all other meshing commands which place nodes using smoothing algorithms to optimize element quality, node locations for the radialmesh command are calculated based on the input radii, angles, and offsets. In addition, the radialmesh command does not mesh existing geometry. Rather, it creates a mesh based on the input parameters, after which a mesh-based geometry is created to fit the free mesh.

The radialmesh command requires input for the 3 coordinate directions (Z, radial, angular). The number of blocks in each direction is specified with the numZ, numR, and numA values in the command. Each block forms a new volume in the final mesh. All bodies in the mesh are merged to form a conformal mesh between blocks.

The Radialmesh command can create meshes which span any angle greater than 0.0 up to 360 degrees. In addition, meshes can model either a tri-section (see Figure 1), or a non-trisection mesh (see Figure 2).

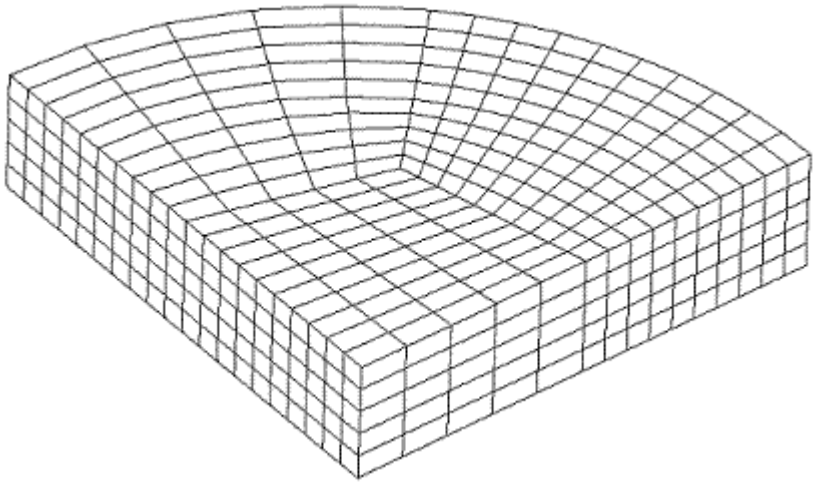


Figure 1. Tri-section Radialmesh

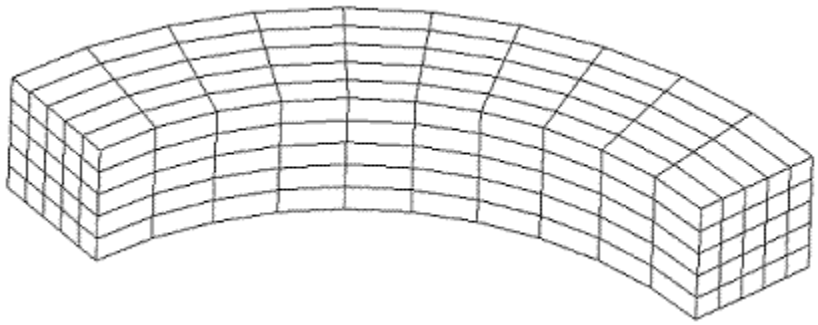


Figure 2. Non-tri-section Radialmesh

The command to generate the mesh in Figure 1 is:

```
create radialmesh \
    numZ 1 zblock 1 1 interval 5 \
    numR 3 trisection rblock 1 2 interval 5 \
            rblock 2 3 interval 5 \
            rblock 3 4 interval 5 \
    numA 1 span 90 ablock 1 interval 10
```

The command to generate the mesh in Figure 2 is:

```
create radialmesh \
    numZ 1 zblock 1 1 interval 5 \
    numR 1 initial radius 3 rblock 1 4 interval 5 \
    numA 1 span 90 ablock 1 interval 10
```

A mesh can span an entire 360 degrees by using the “full360” keyword. For example, the mesh in Figure 3 was generated with the following command:

```
create radialmesh numZ 1 zblock 1 1 interval 5 \
  numR 3 trisection rblock 1 1 interval 5 \
      rblock 2 2 interval 5 \
      rblock 3 3 interval 5 \
  numA 5 full360 span ablock 1 interval 5 \
      ablock 2 interval 5 \
      ablock 3 interval 5 \
      ablock 4 interval 5
```

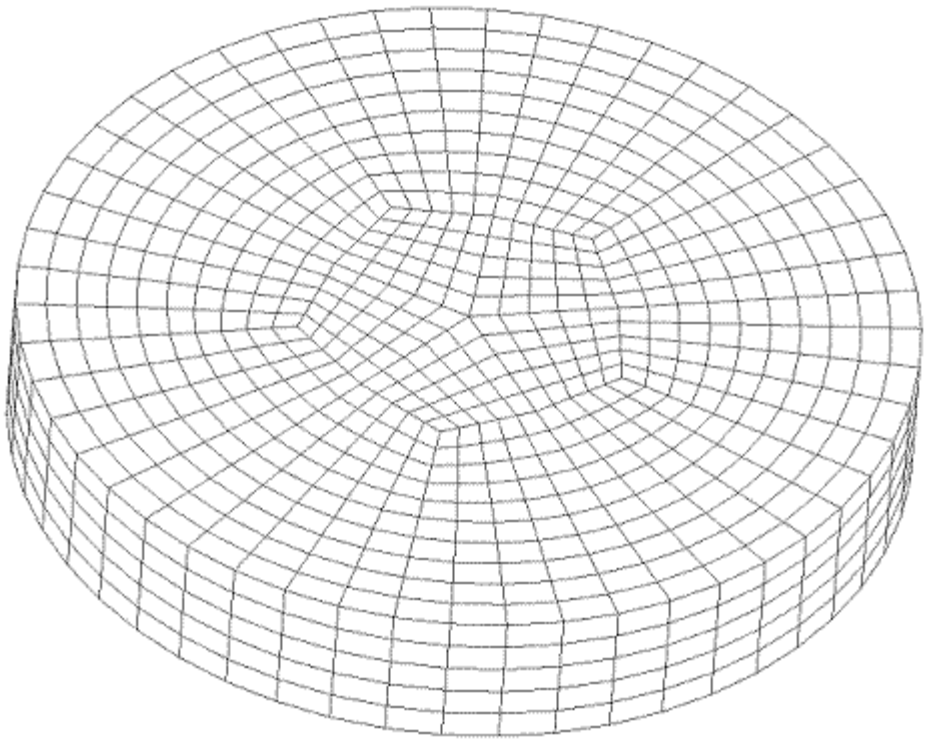


Figure 4. Radialmesh using full360 option

After the mesh is generated, the radialmesh command fits the mesh with mesh based geometry. The surfaces created to fit the mesh are given special names according to their location on the geometry. To see the names of the surfaces, issue the command **label surface name** after creating a radialmesh. Also, if you create a tri-section mesh, the edges on the center axis are given names. To see these names issue the command **label curve name** after creating a trisection Radialmesh.

The user can control the number of intervals and the spacing of these intervals using the optional parameters in each rblock, zblock and ablock. There are 11 combinations that these can be combined as listed below:

- **Interval Only**- Example: "interval 5." The block will be meshed with 5 equally spaced intervals.
- **First Size Only**- Example: “first size 2.5.” The block will be meshed with intervals of approximately 2.5 in length. The total number of intervals is internally calculated and depends on the overall block length.
- **Fraction Only**- Example: “fraction 0.3333.” The block will be meshed with intervals approximately 0.3333*overall block length.
- **Interval and Bias**- Example: “interval 5 bias 1.5.” There will be 5 intervals on the block, which each interval being 1.5 times the previous one. The length of each interval is calculated internally.
- **Interval and Fraction**- Example: “interval 5 fraction 0.25.” There will be 5 intervals on the block, the first being .25 of the length of the block with the remaining decreasing in size.
- **Interval and First Size**- Example: “interval 5 first size 0.2.” There will be 5 intervals on the block, the first being 0.2 in length. The remaining intervals will increase or decrease to fill the blocks length.
- **First Size and Last Size**- Example: “first size 0.2 last size 0.4.” The first interval will be 0.2 in length. The last interval will be 0.4 in length. The total number of intervals is internally calculated to allow for transition between the 2 specified sizes.
- **First Size and Bias**- Example “first size 0.2 bias 0.85.” The first interval will be 0.2 in length and the remaining intervals will scale by a factor of 0.85 from one to the next until the block is filled. The total number of intervals is internally calculated and depends on the overall block length.

- **Fraction and Bias**- Example “fraction 0.25 bias 1.25.” The first interval will be 0.25 of the overall block length and the remaining intervals will scale by a factor of 1.25 from one to the next until the block is filled. The total number of intervals is internally calculated and depends on the overall block length.

- **Interval and Last Size**- Example: “last size 1.5 interval 5.” The last interval will be 1.5 in length. The remaining intervals will scale up or down to fit 5 intervals in the block.

- **Last Size and Bias**- Example: “last size 2.0 bias 1.1.” The last interval will be 2.0 in length. The remaining intervals will scale by 1.1 until the block is filled. The total number of intervals is internally calculated and depends on the overall block length.

Figure 4 shows an example of a bias spaced mesh with the following command:

```
create radialmesh numZ 2 zblock 1 1 first size 0.2 \
        zblock 2 10 first size 0.2 last size 1.0 \
    numR 3 trisection rblock 1 1 interval 5 \
        rblock 2 2 first size .25 \
        rblock 3 5 first size .25 bias 2.0 \
    numA 1 span 90 ablock 1 interval 5
```

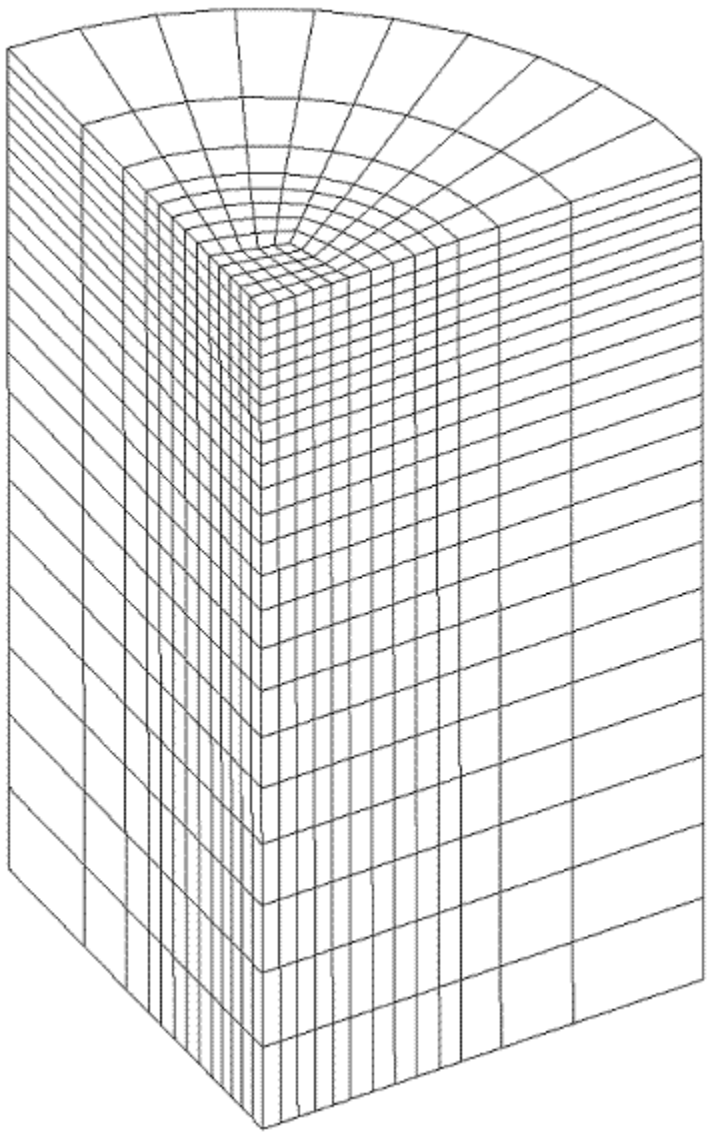


Figure 4. Radialmesh created with biased spacing

Dice

Applies to: Curves, Surfaces, Volumes

Summary: Refinement algorithm for splitting coarse quads and hexes into smaller entities of the same type.

Syntax:

{Curve|Surface|Volume} <range> Scheme Dice

Related Commands:

{Curve|Surface|Volume} <range>

Initialize Dicer{Curve|Surface|Volume} <range>

DicerSheet Interval <interval> {Curve|Surface|Volume} <range>

DicerSheet Interval Size <size>DicerSheet <id> interval <interval>

DicerSheet Default Interval <interval>

Replace Mesh {Surface|Volume|Group} <range>

Set Node Constraint [ON|off]Delete Fine Mesh {Volume|Surface|Curve} <range> [Propagate]

DicerSheet <id> Bias <value> Start Node <id>

Refining a Mesh with Dicing

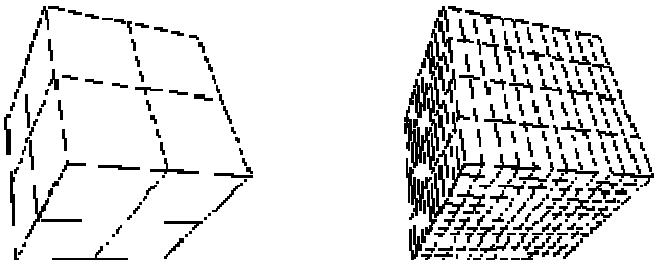
The commands used to dice a mesh are very similar to those used to generate a mesh with other meshing schemes. To refine a mesh with dicing, follow these steps:

1. Set the mesh scheme to Dice for each entity to be diced, using a command such as Volume 1 Scheme Dice.
2. Set the interval on the entity, using a command such as Volume 1 Interval 3. This will set the refinement interval for the specified volumes. For a definition of a refinement interval , see the detailed discussion below.
3. Mesh the entity, using a command such as Mesh Volume 1. This will generate a fine mesh, but will not apply it to the geometry (the view of the mesh in the graphics window will not change).
4. Replace the course mesh with the fine mesh, using a command such as Replace Mesh Volume 1. This will apply the fine mesh to the geometry, and will delete the previously existing coarse mesh. The changes in the mesh will be visible in graphics window.

Detailed Discussion:

Occasionally, it is more convenient to mesh a volume in two stages, first generating a coarse mesh, and then converting the coarse mesh to a fine mesh. The method used to convert a coarse hex mesh to a fine hex mesh is known as dicing.

Dicing (Melander, 97) replaces each hex in a coarse mesh with a grid of smaller hexes. The grid is generated by cutting the hex any number of times along each of its three primary axes. The number of fine hexes in the grid depends on the number of cuts in each direction. The number of cuts along any of the hex's three primary axes is known as the refinement interval of that axis (also known as the dicersheet interval). For example, a hex with a refinement interval of 2 in each direction will be replaced by a grid of 8 smaller elements. A simple example is shown in the following figure.



Simple Dice Example

Dicing may also be performed on a quad mesh. The result is a grid of quads replacing each coarse quad element.

In order for the resulting fine mesh to be conformal, groups of coarse mesh edges must have the same refinement interval. Each group of dependent edges is known as a dicersheet. Dicersheets often include edges from several surfaces and volumes, so dependencies may propagate throughout the mesh. Dicersheets are maintained automatically and enforce refinement interval dependencies.

Extended Dicing Commands

In addition to the steps described above, an alternative set "extended" commands may be used to dice a mesh. These steps correspond more closely to the internal process CUBIT uses to refine the mesh.

1. **Initialize the dicer**

   Before dicing may be carried out, the dicer must first be initialized. This will create the necessary internal data needed to enforce constraints and correctly generate and store the fine mesh. To initialize the dicer for a given entity, use the command <Entity_List> Initialize Dicer. This command will cause all appropriate internal data to be generated. If there are dependencies between any of the specified entities, or any entity for which the dicer has already been initialized, those dependencies will automatically be reflected in the internal data via dicer sheets.

2. **Set refinement intervals**

   After the dicer has been initialized, refinement intervals should be set. This will determine the number of fine edges replacing each coarse edge in a given dicer sheet, ultimately determining the number of fine elements that will replace each coarse element. The refinement interval must be a positive integer, 1 or greater. A refinement interval of 1 will leave the coarse edges unchanged, replacing 1 coarse edge with 1 fine edge.

   Refinement intervals may be set on a geometric entity, on individual dicer sheets, or using a default value for all dicer sheets, using the commands:

   {Volume|Surface|Curve} <range> DicerSheet Interval <interval>

   DicerSheet <id> Interval <interval>

   DicerSheet Default Interval <interval>

   The default dicersheet interval is two.

   It is also possible to set a dicersheet interval size by using the command:

   {Volume|Surface|Curve} <range> DicerSheet Interval Size <size>

   One additional command allows biasing of dicersheets. A start node id, which must be found in the dicersheet, is input to determine from which side of the dicersheet to begin the bias.

DicerSheet <id> Bias <value> Start Node <id>

3. **Perform the dicing**

   Initializing the dicer for an entity will set the mesh scheme for that entity to Dice. Once the scheme has been set, the coarse mesh can be used to create the fine mesh using the command

   Mesh {Volume|Surface|Curve} <range>

   The fine mesh will be generated and will exist in memory, but at this point will not be applied to the entity that was diced.

4. **Replace the coarse mesh with the fine mesh.**

   Once the fine mesh exists in memory, you may replace the coarse mesh with the fine mesh with the command

   Replace Mesh {Volume|Surface} <range>

   This command works only with surfaces and volumes. Each coarse element will be replaced with its grid of fine elements. As a result, the mesh on any child entities will also be replaced. In other words, replacing the mesh of a volume will also replace the mesh on each of that volume's surfaces and curves.

   NOTE: You may find it difficult to view the fine mesh, until after you have completed the replace mesh step.

   As a coarse mesh is replaced, any coarse elements that are still needed by another portion of the mesh will not be destroyed. For example, assume that two volumes have been merged and shared a surface. If both volumes are meshed, and the mesh on one volume is then replaced, the shared coarse surface mesh will still exist because it is needed by the other volume. At this point, the surface mesh is in an ambiguous state, simultaneously containing coarse and fine elements. If the second volume is then diced and its mesh is replaced, the coarse mesh on the shared surface will then be deleted and the fine mesh will be conformal between the two volumes.

Additional Dicing Commands

Several utilities have been developed to assist the user during the refinement process.

Constraining Nodes to Geometry:

The user can control whether refinement nodes of surface and curve meshes get moved to the geometry, or whether their positions remain as a straight-line interpolation between coarse nodes, via the following command:

Set Node Constraint {on|off}

If Node Constraint is on, which is the default, then nodes are constrained to lie on the geometry.

Deleting a Fine Mesh

The fine nodes generated by the Dicer may be deleted using the command

Delete Fine Mesh {geom_list} [Propagate]

This command only works before using the Replace Mesh command. Any fine mesh entities that rely on the deleted fine nodes are also deleted. For example, if the fine nodes on a surface are deleted, the fine mesh of any attached volume is deleted along with the nodes on the surface. If the optional Propagate keyword is used, the fine mesh will be deleted from any child entities as well.

Interaction with Dicer Sheets

Dicer sheets can be drawn, picked, highlighted, and listed, like other entities in the CUBIT model.

HTet

Applies to: Volumes

Summary: Converts an existing hex mesh into a conforming tetrahedral mesh.

Syntax:

HTet Volume <range> {UNSTRUCTURED | structured}

Discussion:

Unlike other meshing schemes in this section, The HTet command requires an existing hexahedral mesh on which to operate. Rather than setting a meshing scheme for use with the mesh command, the HTet command works after an initial hex mesh has been generated.

Two methods for decomposing a hex mesh into tetrahedra are available. Set the method to be used with the optional arguments unstructured and structured. The unstructured method is the default. Figure 1 shows the difference between the two methods:

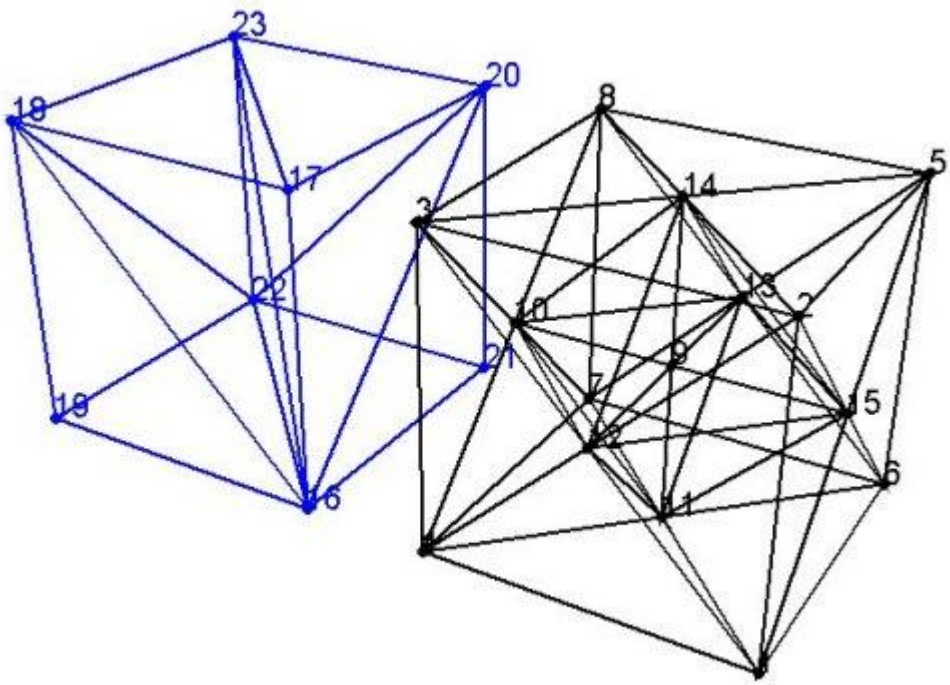


Figure 1. Left: Unstructured method creates 6 tets per hex. Right: Structured method creates 27 tets per hex

Unstructured

This method creates 6 tetrahedra for every hexahedra. No new nodes will be generated. The orientation of the 6 hexahedra will be based upon the element node numbering, as a result orientations may change if node numbering changes. This method is referred to as unstructured because the number of tetrahedra adjacent each node will be relatively arbitrary in the final mesh. Tetrahedral element quality is generally sufficient for most applications, however the user may want to verify quality before performing analysis.

Structured

With this approach, 27 tetrahedra are generated for every hexahedra in the mesh. This method adds a node to each face of the hex and one to the interior. Although this method generates significantly more elements, the orientation and quality of the resulting tetrahedra are more consistent. Each previously existing interior node in the mesh will have the same number of adjacent tetrahedra.

QTri

Applies to: Surfaces

Summary: Meshes surfaces using a quadrilateral scheme, then converts the quadrilateral elements into triangles.

Syntax:

Surface <range> Scheme Qtri [Base Scheme quad_scheme>]

QTri Surface <range>

set qtri split [2|4]

Discussion:

QTri is used to mesh surfaces with triangular elements. The surface is, first, meshed with the quadrilateral scheme, and, then, the generated quads are split along a diagonal to produce triangles. The first command listed above sets the meshing scheme on a surface to QTri. The second form sets the scheme and generates the mesh in a single step.

In the first command, the user has the option of specifying the underlying quadrilateral meshing scheme using the base scheme <quad_scheme> option. If no base scheme is specified, CUBIT will automatically select a scheme. For non-periodic surfaces, the base scheme will be set to scheme pave. For periodic surfaces, the base scheme will be set to scheme map.

Generally, the second command, Qtri Surface <range>, is used on surfaces that have already been meshed with quadrilaterals. If, however, this command is used on a surface that has not been meshed, a base scheme will automatically be selected using CUBIT's auto-scheme capabilities. The user can over-ride this selection by specifying a quadrilateral meshing scheme prior to using the qtri command (using the Surface <range> Scheme <quad_scheme> command).

In addition to the default 2 tris per quad, the set qtri split command may alter the QTri scheme so that it will split the quad into 4 triangles per quad. Where the 4 option is used, an additional mesh node is placed at the centroid of each quad.

Also, the QTri scheme is used in the TriMesh command as a backup to the TriAdvance triangle meshing scheme.

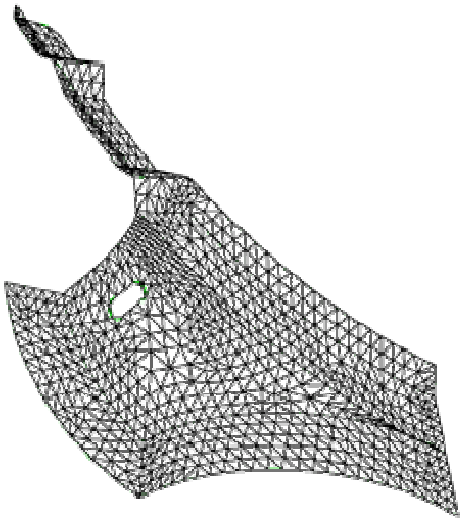


Figure 1. Surface meshed with scheme QTri

THex

Applies to: Volumes

Summary: Converts a tetrahedral mesh into a hexahedral mesh.

Syntax:

THex Volume <range>

Discussion:

The THex command splits each tetrahedral element in a volume into four hexahedral elements, as shown in Figure 1. This is done by splitting each edge and face at its midpoint, and then forming connections to the center of the tet.

When THexing merged volumes, all of the volumes must be THexed at the same time, in a single command. Otherwise, meshes on shared surfaces will be invalid. An example of the THex algorithm is shown in Figure 2.

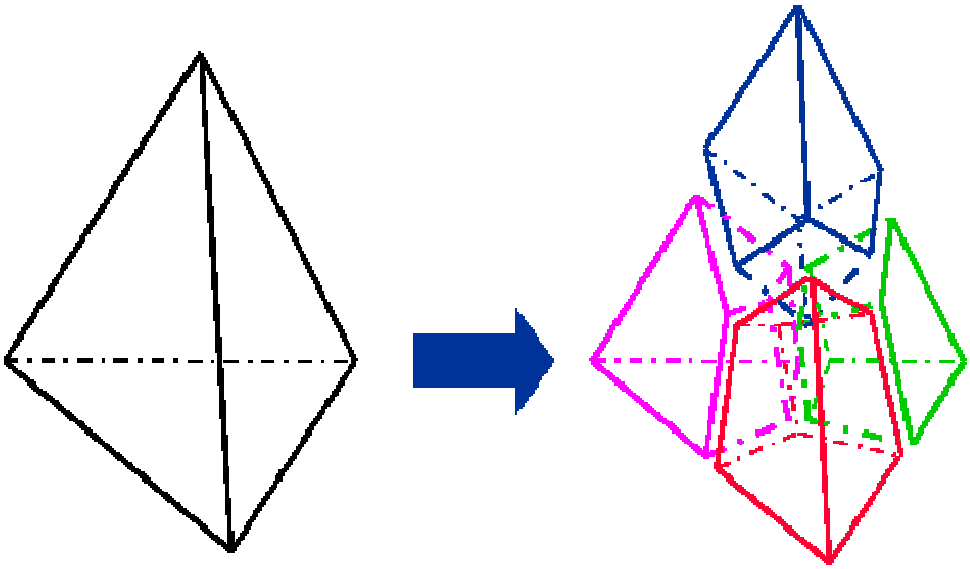


Figure 1. Conversion of a tetrahedron to four hexahedra, as performed by the THex algorithm.

.

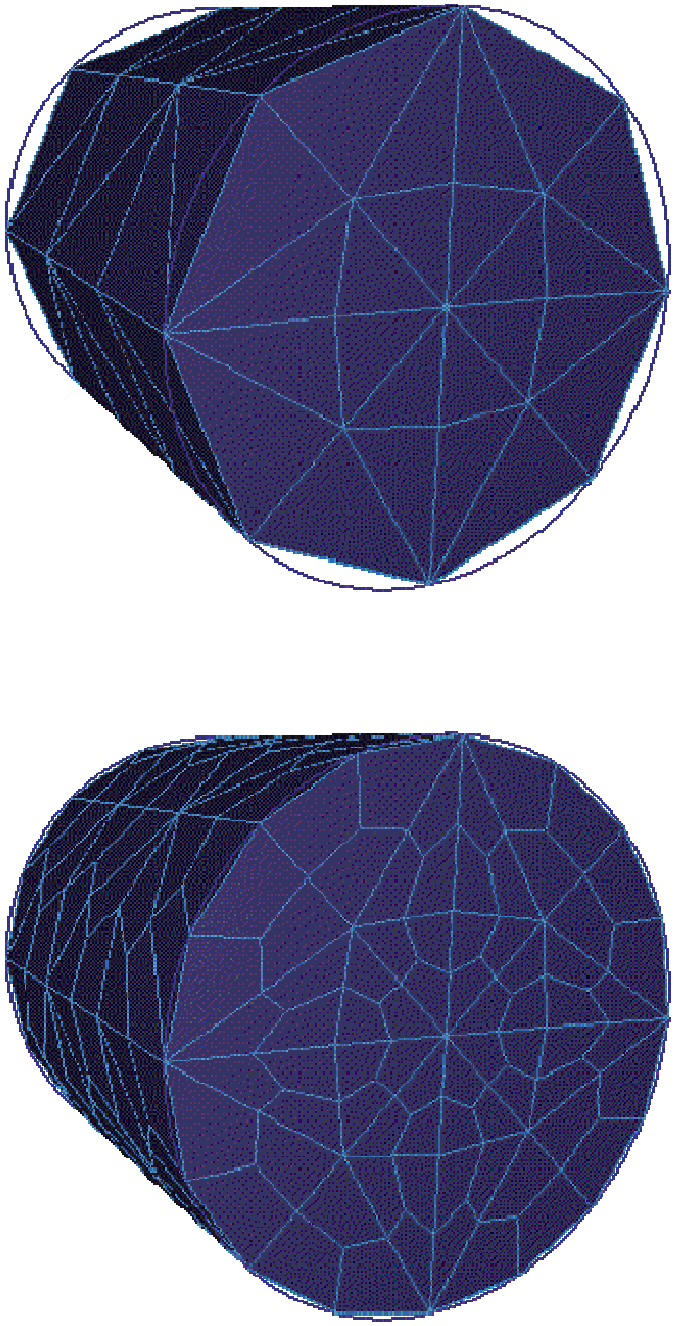


Figure 2. A cylinder before and after the THex algorithm is applied.

Copying a Mesh

Applies to: Curves, Surfaces, Volumes

Summary: Copies the mesh from one entity to another

Syntax:

Curve <range> Scheme Copy source Curve <range> [Source Percent [<percentage> | auto]] [Source [combine|SEPARATE]] [Target [combine|SEPARATE]] [Source Vertex <id_range>] [Target Vertex <id_range>]]

Surface <range> Scheme Copy [Source Surface] <id> [[Source Curve <id> Target Curve <id>] [Source Vertex <id> Target Vertex <id>] [Nosmoothing]

Volume <range> Scheme Copy [Source Volume] <id> [[Source Surface <id> Target Surface <id>] [Source Curve <id> Target Curve <id>] [Source Vertex <id> Target Vertex <id>]][Nosmoothing]

Copy Mesh Curve <curve_id_range> Onto Curve <curve_id_range> [Source Node <starting node id> <ending node id>] [Source Percent [<percentage>|auto]] [Source [combine|SEPARATE]] [Target [combine|SEPARATE]] [Source Vertex <id_range>] [Target Vertex <id_range>]

Copy Mesh Surface <surface_id> Onto Surface <surface_id> [Source Face <id_range>] [Source Node <id> Target Node <id>] [Source Edge <id> Target Edge <id>] [Source Vertex <id> Target Vertex <id>] [Source Curve <id> Target Curve <id>] [Nosmoothing]

Copy Mesh Volume <volume_id> Onto Volume <volume_id> [Source Vertex <vertex_id> Target Vertex <vertex_id> [Source Curve <curve_id> Target Curve <curve_id>] [Nosmoothing]

Related Commands:

Set Morph Smooth {on | off}

Discussion:

If the user desires to copy the mesh from a surface, volume, curve, or set of curves that has already been meshed, the copy mesh scheme can be used. Note that this scheme can be set before the source entity has been meshed; the source entity will be meshed automatically, if necessary, before the mesh is copied to the target entity. The user has the option of providing orientation data to specify how to orient the source mesh on the target entity. For example, when copying a curve mesh, the user can specify which vertex on the source (the source vertex) gets copied to which vertex on the target (the target vertex). If you need to reference mesh entities for the copy, use the **Copy Mesh** commands. If no orientation data is specified, or if the data is insufficient to completely determine the orientation on the target entity, the copy algorithm will attempt to determine the remaining orientation data automatically. If conflicting, or inappropriate, orientation data is given, the algorithm attempts to discard enough information to arrive at a proper mesh orientation.

Curve mesh copying has certain options that allow the copying of just a section of the source curves' mesh. These options are accessed through the extra keyword options. The percentoption allows the user to specify that a certain percentage of the source mesh be copied--in this context the auto keyword means that the percentage will be calculated based on the ratio of lengths of the source and target curves. The combine and separate keywords relate to how the command line options are interpreted. If the user wishes to specify a group of target curves that will each receive an identical copy of a source mesh, then the target separate option should be used (this is the default). If, however, the user wishes the source mesh to be spread out along the range of target curves, then the target combine option should be used. The source curves are treated in a similar fashion.

Volume mesh copying depends on the surface copying scheme. Because of this, the target volume must not have any of its surfaces meshed already.

Because of how the copying algorithm works, the target mesh might not be an exact copy of the source mesh. This happens because of the effects of smoothing. If an exact copy is required, there are two possible solutions. The first option is useful when the source and target surfaces or volumes are exact matches. If this criterion is met, the user may specify the Nosmoothing option. That will disable any smoothing of the mesh on the target surface and thereby providing an exact copy of the mesh. The second option is useful if the source and target surfaces are not identical. In this case the user may set the morph smoothing flag on, which will activate a special smoother that will match up the meshes as closely as possible.

Mirroring a Mesh

Applies to: Surfaces

Summary: Mirrors the mesh from one surface to another

Syntax:

Surface <range> Scheme Mirror [Source Surface <id> [Source Vertex <id> Target Vertex <id>]] [Nosmoothing]

Mirror Mesh Surface <surface_id> Onto Surface <surface_id> [Source Vertex <id> Target Vertex <id> Source Curve <id> Target Curve <id> Source Node <id> Target Node <id>] [Nosmoothing]

Discussion:

The mirror scheme is very similar to the copy scheme. In order to understand what is changed, a discussion of the copy command is in order. Depending on what the user enters for the copy scheme, the resulting mesh might be oriented one of two ways. For example, if the user entered:

Surface 1 scheme copy source surface 2 source vertex 5 target vertex 1

then the algorithm would match vertex 1 with vertex 5, but then would have to make a guess about how to match the curves. Lacking other pertinent data, the match will be a direct match, as is shown in the following figure:

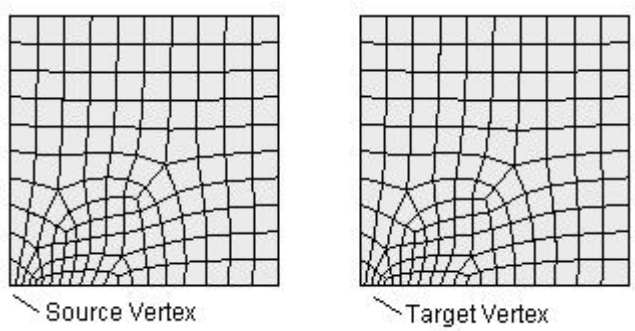


Figure 1. Surface 1 copied onto surface 2

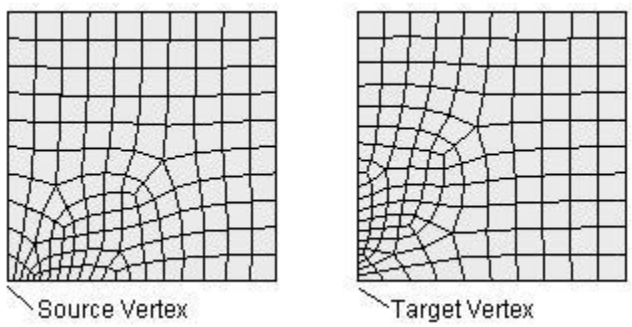


Figure 2. Surface 1 mirrored onto surface 2

This default matching can be changed by specifying more information for matching, or the user can specify scheme mirror. The mirror scheme sets up the copying information in such a way as to reverse the default orientation of the target mesh, as is shown in the above figure (right).

There are times when the resulting mesh may not match the original mesh exactly due to smoothing. Using the nosmoothing option will ensure that the resulting mesh matches the original mesh exactly.

The alternate form of the command copies the mesh immediately instead of setting a scheme first. This form of the command can also use curves and mesh entities as references.

Automatic Scheme Selection

- Default Scheme Selection
- Automatic Scheme Selection General Notes
- Surface Auto Scheme Selection
- Volume Auto Scheme Selection

For volume and surface geometries the user may allow CUBIT to automatically select the meshing scheme. Automatic scheme selection is based on several constraints, some of which are controllable by the user. The algorithms to select meshing schemes will use topological and geometric data to select the best quad or hex meshing tool. Auto scheme selection will not select tet or tri meshing algorithms. The command to invoke automatic scheme selection is:

{geom_list} Scheme Auto

Specifically for surface meshing, interval specifications will affect the scheme designation. For this reason it is recommended that the user specify intervals before calling automatic scheme selection. If the user later chooses to change the interval assignment, it may be necessary to call scheme selection again. For example, if the user assigns a square surface to have 4 intervals along each curve, scheme selection will choose the surface mapping algorithm. However if the user designates opposite curves to have different intervals, scheme selection will choose paving, since this surface and its assigned intervals will not satisfy the mapping algorithm's interval constraints. In cases where a general interval size for a surface or volume is specified and then changed, scheme selection will not change. For example, if the user specified an interval size of 1.0 a square 10X10 surface, scheme selection will choose mapping. If the user changes the interval size to 2.0, mapping will still be chosen as the meshing scheme from scheme selection. If a mesh density is not specified for a surface, a size based on the smallest curve on the surface will be selected automatically.

Default Scheme Selection

If the user does set a scheme for a particular entity and chooses to mesh the entity, CUBIT will automatically run the auto scheme selection algorithm and attempt to set a scheme. In cases where the auto scheme selection fails to choose a scheme, the meshing operation will fail. In this case explicit specification of the meshing scheme and/or further geometry decomposition may be necessary.

The default scheme selection in CUBIT, unless otherwise set, will attempt to set a quadrilateral or hexahedral meshing scheme on the entity. If tet or tri meshing will always be the desired element shape, the following command can be used:

set default element [tet|tri|HEX|QUAD|none]

Setting the default element to **tet** or **tri** will bypass the auto scheme selection and always use either the triadvance or tetmesh schemes if the scheme has not otherwise been set by the user. The default settings of **quad** or **hex** will use the automatic scheme selection.

Previous functionality of CUBIT used a default scheme of map and interval of 1 for all surface and volume entities. For backwards compatibility and if this behavior is still desired, the **none** option may be used on the **set default element** command.

Auto Scheme Selection General Notes

In general, automatic scheme selection reduces the amount of user input. If the user knows the model consists of 2.5D meshable volumes, three commands to generate a mesh after importing or creating the model are needed. They are:

volume all size <value>

volume all scheme auto

mesh volume all

The non-trivial, academic model shown in the following figure meshed using automatic scheme selection. was meshed using these three commands (part of the model is not shown to reveal the internal structure of the model).

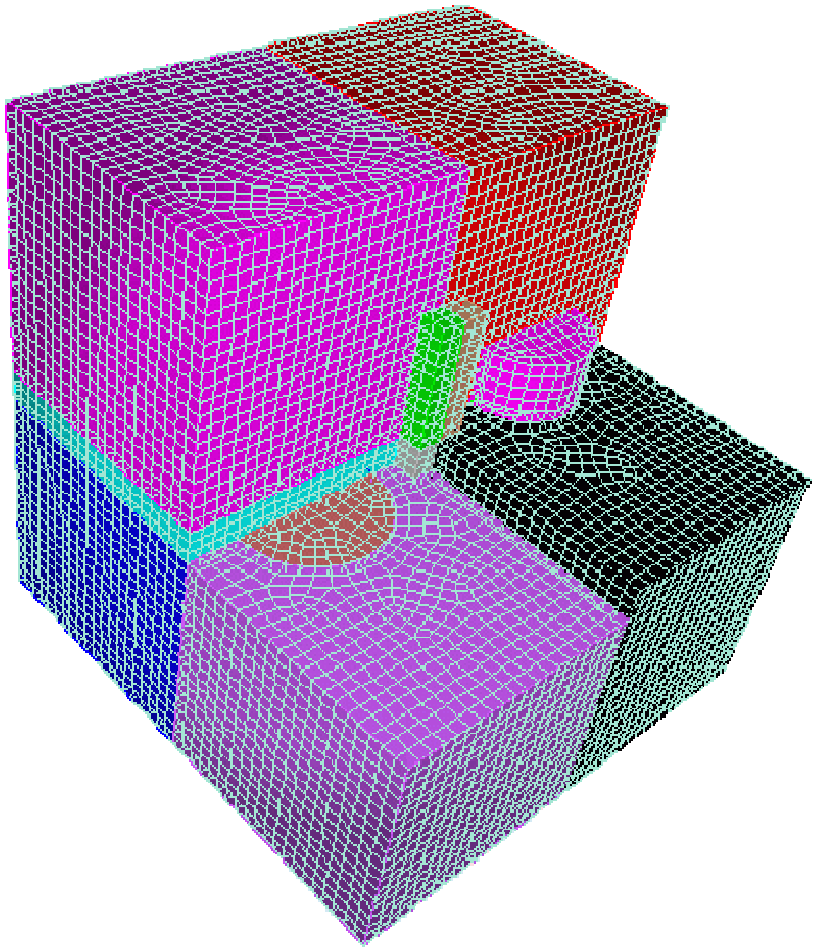


Figure 1. Non-trivial model meshed using automatic scheme selection

Scheme Firmness

Meshing schemes may be selected through three different approaches. They are: default settings, automatic scheme selection, and user specification. These methods also affect the scheme firmness settings for surfaces and volumes. Scheme firmness is completely analogous to interval firmness.

Scheme firmness can be set explicitly by the user using the command

{geom_list} Scheme {Default | Soft | Hard}

Scheme firmness settings can only be applied to surfaces and volumes.

This may be useful if the user is working on several different areas in the model. Once she/he is satisfied with an area's scheme selection and doesn't want it to change, the firmness command can be given to hard set the schemes in that area. Or, if some surfaces were hard set by the user, and the user now wants to set them through automatic scheme selection then she/he may change the surface's scheme firmness to soft or default.

Surface Auto Scheme Selection

Surface auto scheme selection (White, 99) will choose between Pave, Submap, Triprimitive, and Map meshing schemes, and will always result in selecting a meshing scheme due to the existence of the paving algorithm, a general surface meshing tool (assuming the surface passes the even interval constraint).

Surface auto scheme selection uses an angle metric to determine the vertex type to assign to each vertex on a surface; these vertex types are then analyzed to determine whether the surface can be mapped or submapped. Often, a surface's meshing scheme will be selected as Pave or Triprimitive when the user would prefer the surface to be mapped or submapped. The user can overcome this by several methods. First, the user can manually set the surface scheme for the "fuzzy" surface. Second, the user can manually set the "vertex types" for the surface. Third, the user can increase the angle tolerance for determining "fuzziness." The command to change scheme selection's angle tolerances is:

[Set] Scheme Auto Fuzzy [Tolerance] {value} (value in degrees)

The acceptable range of values is between 0 and 360 degrees. If the user enters 360 degrees as the fuzzy tolerance, no fuzzy tolerance checks will be calculated, and in general mapping and submapping will be chosen more often. If the user enters 0 degrees, only surfaces that are "blocky" will be selected to be mapped or submapped, and in general paving will be chosen more often.

Volume Auto Scheme Selection

When automatic scheme selection is called for a volume, surface scheme selection is invoked on the surfaces of the given volume. Mesh density selections should also be specified before automatic volume scheme selection is invoked due to the relationship of surface and volume scheme assignment.

Volume scheme selection chooses between Map, Submap and Sweep meshing schemes. Other schemes can be assigned manually, either before or after the automatic scheme selection.

Volume scheme selection is limited to selecting schemes for 2.5D geometries, with additional tool limitations (e.g. Sweep can currently only sweep from several sources to a single target, not multiple targets); this is due to the lack of a completely automatic 3D hexahedral meshing algorithm. If volume scheme selection is unable to select a meshing scheme, the mesh scheme will remain as the default and a warning will be reported to the user.

Volume scheme selection can fail to select a meshing scheme for several reasons. First, the volume may not be mappable and not 2.5D; in this case, further decomposition of the model may be necessary. Second, volume scheme selection may fail due to improper surface scheme selection. Volume schemes such as Map, Submap, and Sweep require certain surface meshing schemes, as mentioned previously.

Meshing the Geometry

After assigning interval or sizing attributes to a geometric entity and a meshing scheme is applied, the geometry is ready to be meshed. To mesh a geometric entity, use the command:

Mesh <entity> <id_range> [GLOBAL | Individual]

The **<entity>** to be meshed may be any one of the following:

Body
Volume
Surface
Curve
Vertex

The **Global** and **Individual** options affect how the constraints are gathered for interval matching. With the Global option, the interval constraint equations are calculated from all entities in the entity list. The Individual option calculates the interval constraint equations from each entity individually. The Global option is the default.

Default Scheme and Interval Selection

If either interval settings or schemes have not already been set on the entities being meshed, CUBIT will do its best to automatically set one or both of these attributes. See Auto Scheme Selection and Auto Specification of Intervals for a description of how CUBIT chooses these attributes. In cases where the automatic scheme selection algorithm fails to select a scheme for the geometry, the meshing operation will fail. In this case explicit specification of the meshing scheme and/or further geometry decomposition may be necessary.

Remeshing a Volume

The mesh generation is frequently an iterative process of meshing, deleting the mesh and remeshing. The remesh command is a convenient tool to bypass the mesh deletion process. To remesh a volume use the following command:

Remesh Volume <id_range>

Remeshing a Swept Volume Mesh

This command is especially useful when using the sweep scheme. When a sweep scheme is applied to the volume, it will delete the target surface mesh on a volume with one of the sweeping schemes and then remesh the volume. It is useful when changing between sweep smooth options as in the following example below.

volume 1 scheme sweep
mesh volume 1

At this stage, the user may discover that poor quality elements may have been generated. The use could then do the following:

volume 1 sweep smooth winslow
remesh volume 1

At this point, volume is remeshed using the sweep smooth winslow option

Continuing Meshing After a Mesh Failure

Frequently when meshing large assemblies containing a number of volumes, the mesh command can be applied to a group of volumes with the same mesh command. Typically, if a mesh failure is detected, the meshing operation will continue to mesh the remaining volumes specified at the command line. The following command permits the user to override this feature to discontinue meshing additional volumes and return to the command line immediately after a mesh failure is detected:

set continue meshing [ON|off]

The default for this command is **ON**.

Turning this setting **OFF** is useful when meshing assemblies where a meshing failure of one volume would adversely affect the meshing of adjoining volume(s). This occurs frequently when meshing a sweep group using the sweep scheme.

Mesh Quality Assessment

- Metrics for Triangular Elements
- Metrics for Quadrilateral Elements
- Metrics for Tetrahedral Elements
- Metrics for Hexahedral Elements
- Mesh Quality Command Syntax
- Mesh Quality Example Output
- Automatic Mesh Quality Assessment
- Controlling Mesh Quality
- Coincident Node Check

The `quality' of a mesh can be assessed using several element quality metrics available in CUBIT. Information about the CUBIT quality metrics can be obtained from the command

Quality Describe { hex | hexahedral | tet | tetrahedral | face | quad | quadrilateral | tri | triangular }

which gives data on the quality metrics for each of the above element types. The following items discuss the mesh quality assessment capabilities in CUBIT:

Metrics for Triangular Elements

The metrics used for triangular elements in CUBIT are summarized in the following table:

Function Name	Dimension	Full Range	Acceptable Range	Reference
Element Area	L^2	0 to inf	None	1
Maximum Angle	degrees	60 to 180	60 to 90	1
Minimum Angle	degrees	0 to 60	30 to 60	1
Condition No	L^0	1 to inf	1 to 1.3	2
Scaled Jacobian	L^0	-1.155 to 1.55	0.5 to 1.155	2
Relative Size	L^0	0 to 1	0.25 to 1	3
Shape	L^0	0 to 1	0.25 to 1	3
Shape and Size	L^0	0 to 1	0.25 to 1	3
Distortion	L^2	-1 to 1	0.6 to 1	4

Approximate Triangular Quality Definitions:

Element Area: (1/2) * Jacobian at corner node

Maximum Angle: Maximum included angle in triangle

Minimum Angle: Minimum included angle in triangle

Condition No. Condition number of the Jacobian matrix

Scaled Jacobian: Minimum Jacobian divided by the lengths of 2 edge vectors

Relative Size: Min(J, 1/J), where J is determinant of weighted Jacobian matrix

Shape: 2/Condition number of weighted Jacobian matrix

Shape & Size: Product of Shape and Relative Size

Distortion: {min(|J|)/actual area}*parent area, parent area = 1/2 for triangular element

Comments on Algebraic Quality Measures

Relative Size, Shape, and Shape & Size are algebraic metrics, which have well behaved properties. Cubit encourages the use of these metrics over other metrics. These metrics are referenced to an ideal element which, in the case of triangular elements, is an equilateral triangle. Thus deviations from an equilateral triangle are measured in various ways by the algebraic metrics.
Relative size measures the size of the element vs. the size of reference element. If the element is twice or one-half the size of the reference element, the relative size is one-half. By default, the size of the reference element is the average size of all the elements that the quality command is currently evaluating.

The shape and size metric measures how both the shape and relative size of the element deviate from that of the reference element.

References for Triangular Quality Measures

1. Traditional.
2. Knupp, 2000.
3. P. Knupp, Algebraic Mesh Quality Metrics for Unstructured Initial Meshes, submitted for publication.
4. SDRC/IDEAS Simulation: Finite Element Modeling--User's Guide

Metrics for Quadrilateral Elements

The metrics used for quadrilateral elements in CUBIT are summarized in the following table:

Function Name	Dimension	Full Range	Acceptable Range	Reference
Aspect Ratio	L^0	1 to inf	1 to 4	1
Skew	L^0	0 to 1	0 to 0.5	1
Taper	L^0	0 to +inf	0 to 0.7	1
Warpage	L^0	0 to 1	0.9 to 1.0	NEW
Element Area	L^2	-inf to inf	None	1
Stretch	L^0	0 to 1	0.25 to 1	2
Minimum Angle	degrees	0 to 90	45 to 90	3
Maximum Angle	degrees	90 to 360	90 to 135	3
Condition No.	L^0	1 to inf	1 to 4	4
Jacobian	L^2	-inf to inf	None	4
Scaled Jacobian	L^0	-1 to +1	0.5 to 1	4
Shear	L^0	0 to 1	0.3 to 1	5
Shape	L^0	0 to 1	0.3 to 1	5
Relative Size	L^0	0 to 1	0.3 to 1	5
Shear & Size	L^0	0 to 1	0.2 to 1	5

Shape & Size	L^0	0 to 1	0.2 to 1	5
Distortion	L^2	-1 to 1	0.6 to 1	6

Quadrilateral Quality Definitions

Aspect Ratio: Maximum edge length ratios at quad center

Skew: Maximum |cos A| where A is the angle between edges at quad center

Taper: Maximum ratio of lengths derived from opposite edges

Warpage: Cosine of Minimum Dihedral Angle formed by Planes Intersecting in Diagonals

Element Area: Jacobian at quad center

Stretch: Sqrt(2) * minimum edge length / maximum diagonal length

Minimum Angle: Smallest included quad angle (degrees).

Maximum Angle: Largest included quad angle (degrees).

Condition No. Maximum condition number of the Jacobian matrix at 4 corners

Jacobian: Minimum pointwise volume of local map at 4 corners & center of quad

Scaled Jacobian: Minimum Jacobian divided by the lengths of the 2 edge vectors

Shear: 2/Condition number of Jacobian Skew matrix

Shape: 2/Condition number of weighted Jacobian matrix

Relative Size: Min(J, 1/J), where J is determinant of weighted Jacobian matrix

Shear and Size: Product of Shear and Relative Size

Shape and Size: Product of Shape and Relative Size

Distortion: {min(|J|)/actual area}*parent area, parent area = 4 for quad

Comments on Algebraic Quality Measures

Shape, Relative Size, Shape & Size, and Shear are algebraic quality metrics that apply to quadrilateral elements. Cubit encourages the use of these metrics since they have certain nice properties (see reference 5 below). The metrics are referenced to a square-shaped quadrilateral element, thus deviations from a square are measured in various ways.

Shape measures how far skew and aspect ratio in the element deviates from the reference element.

Relative size measures the size of the element vs. the size of reference element. If the element is twice or one-half the size of the reference element, the relative size is one-half. The reference element for the Relative Size metric is a square whose area is determined by the average area of all the quadrilaterals on the surface mesh under assessment

Shape and size metric measures how both the shape and relative size of the element deviate from that of the reference element.

The SHEAR metric is based on the condition number of the skew matrix. SHEAR is really just an algebraic skew metric but, since the word skew is already used in the list of quad quality metrics, Cubit has chosen to use the word 'shear.'

Shear = 1 if and only if quadrilateral is a rectangle.

The Robinson 'skew' metric equals the ideal (zero) if the quad is a rectangle. It also attains the ideal if the quad is a trapezoid, a kite, or even triangular!

References for Quadrilateral Quality Measures

1. (Robinson, 87)
2. FIMESH code.
3. Unknown.
4. (Knupp, 00)
5. P. Knupp, Algebraic Mesh Quality Metrics for Unstructured Initial Meshes, submitted for publication.
6. 6. SDRC/IDEAS Simulation: Finite Element Modeling--User's Guide

Details on Robinson Metrics for Quadrilaterals

The quadrilateral element quality metrics that are calculated are aspect ratio, skew, taper, element area, and stretch. The calculations are based on metrics described in (Robinson, 87). An illustration of the shape parameters is shown in Figure 1, below. The stretch metric is calculated by dividing the length of the shortest element edge divided by the length of the longest element diagonal.

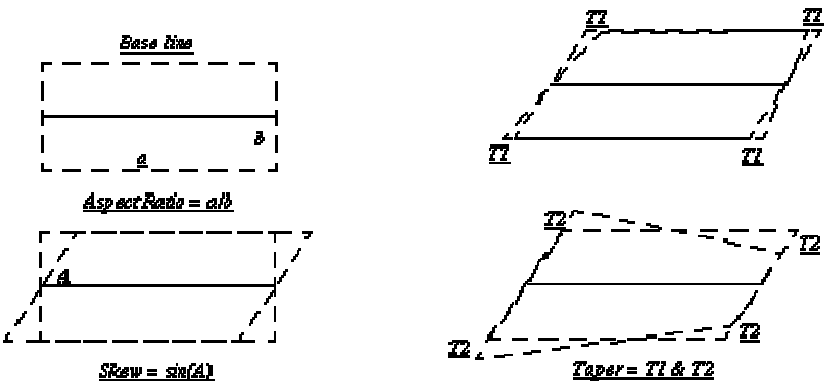


Figure 1. Illustration of Quadrilateral Shape Parameters (Quality Metrics)

Metrics for Tetrahedral Elements

The metrics used for tetrahedral elements in CUBIT are summarized in the following table:

Function Name	Dimension	Full Range	Acceptable Range	Reference
Aspect Ratio Beta	L^0	1 to inf	1 to 3	1
Aspect Ratio Gamma	L^0	1 to inf	1 to 3	1
Element Volume	L^3	-inf to inf	None	1
Condition No	L^0	1 to inf	1 to 3	2
Jacobian	L^3	-inf to inf	None	2
Scaled Jacobian	L^0	-1.414 to 1.414	0.5 to 1.414	2

Shape	L^0	0 to 1	0.2 to 1	3
Relative Size	L^0	0 to 1	0.2 to 1	3
Shape and Size	L^0	0 to 1	0.2 to 1	3
Distortion	L^0	-1 to 1	0.6 to 1	4

Tetrahedral Quality Definitions

Aspect Ratio Beta: CR / (3.0 * IR) where CR = circumsphere radius, IR = inscribed sphere radius

Aspect Ratio Gamma: Srms**3 / (8.479670*V) where Srms = sqrt(Sum(Si**2)/6), Si = edge length

Element Volume: (1/6) * Jacobian at corner node

Condition No.: Condition number of the Jacobian matrix at any corner

Jacobian: Minimum pointwise volume at any corner

Scaled Jacobian: Minimum Jacobian divided by the lengths of 3 edge vectors

Shape: 3/Mean Ratio of weighted Jacobian Matrix

Relative Size: Min(J, 1/J), where J is the determinant of the weighted Jacobian matrix

Shape & Size: Product of Shape and Relative Size Metrics

Distortion: {min(|J|)/actual volume}*parent volume, parent volume = 1/6 for tet

References for Tetrahedral Quality Measures

1. (Parthasarathy, 93)
2. (Knupp, 00)
3. P. Knupp, Algebraic Mesh Quality Metrics for Unstructured Initial Meshes, to appear in Finite Elements for Design
   and Analysis.
4. SDRC/IDEAS Simulation: Finite Element Modeling - User's Guide

Metrics for Hexahedral Elements

The metrics used for hexahedral elements in CUBIT are summarized in the following table:

Function Name	Dimension	Full Range	Acceptable Range	Reference
Aspect Ratio	L^0	1 to inf	1 to 4	1
Skew	L^0	0 to 1	0 to 0.5	1
Taper	L^0	0 to +inf	0 to 0.4	1

Element Volume	L^3	-inf to inf	None	1
Stretch	L^0	0 to 1	0.25 to 1	2
Diagonal Ratio	L^0	0 to 1	0.65 to 1	3
Dimension	L^1	0 to inf	None	1
Condition No.	L^0	1 to inf	1 to 8	5
Jacobian	L^3	-inf to inf	None	5
Scaled Jacobian	L^0	-1 to +1	0.5 to 1	5
Shear	L^0	0 to 1	0.3 to 1	5
Shape	L^0	0 to 1	0.3 to 1	5
Relative Size	L^0	0 to 1	0.5 to 1	5
Shear & Size	L^0	0 to 1	0.2 to 1	5
Shape & Size	L^0	0 to 1	0.2 to 1	5
Distortion	L^0	0 to 1	0.6 to 1	6

Hexahedral Quality Definitions

Aspect Ratio: Maximum edge length ratios at hex center.

Skew: Maximum |cos A| where A is the angle between edges at hex center.

Taper: Maximum ratio of lengths derived from opposite edges.

Element Volume: Jacobian at hex center.

Stretch: Sqrt(3) * minimum edge length / maximum diagonal length.

Diagonal Ratio: Minimum diagonal length / maximum diagonal length.

Dimension: Pronto-specific characteristic length for stable time step calculation. Char_length = Volume / 2 grad Volume.

Condition No. Maximum condition number of the Jacobian matrix at 8 corners.

Jacobian: Minimum pointwise volume of local map at 8 corners & center of hex.

Scaled Jacobian: Minimum Jacobian divided by the lengths of the 3 edge vectors.

Shear: 3/Mean Ratio of Jacobian Skew Matrix

Shape: 3/Mean Ratio of weighted Jacobian Matrix

Relative Size: Min(J, 1/J), where J is the determinant of weighted Jacobian matrix

Shear & Size: Product of Shear and Size Metrics

Shape & Size: Product of Shape and Size Metrics

Distortion: {min(|J|)/actual volume}*parent volume, parent volume = 8 for hex

References for Hexahedral Quality Measures

1. (Taylor, 89)
2. FIMESH code
3. Unknown
4. (Knupp, 00)
5. P. Knupp, Algebraic Mesh Quality Metrics for Unstructured Initial Meshes, to appear in Finite Elements for Design and Analysis.
6. SDRC/IDEAS Simulation: Finite Element Modeling - User's Guide

Mesh Quality Command Syntax

The base command to view the quality of a mesh is the following:

Quality {geom_and_mesh_list} [metric name] [quality options] [filter options]

Where the list contains surfaces and volumes and groups that have been meshed with faces, triangles, hexes, and tetrahedra; the list can also specify individual mesh entities or ranges of mesh entities.

If a specific metric name is given, only that metric or metrics are computed for the specified entities. Note that the metric given must be one which applies to the given entities; quality metrics and which entities which they apply are summarized above.

The metric name can also be more general than a specific metric. Four generalized options for metric name can be used:

Allmetrics: All of the metrics corresponding to the element type of the geom_and_mesh_list will be computed and reported.

Algebraic: All algebraic metrics corresponding to the element type of the geom_and_mesh_list will be computed and reported (e.g., Shape, Shear, Relative Size).

Robinson: All Robinson metrics corresponding to the element type of the geom_and_mesh_list will be computed and reported (e.g., Aspect Ratio, Skew, Taper).

Traditional: All the traditional Cubit metrics corresponding to the element type of the geom_and_mesh_list will be computed and reported (e.g., area, volume, angle, stretch, dimension).

If no metric name is supplied, the default metric is "**Shape**".

Quality Options

The quality options are:

Scope

[Global | Individual]

If the user specifies **individual**, one quality summary is generated for each entity specified on the command line. If the user specifies **global**, or specifies neither, then one quality summary is generated for each mesh element type.

Draw

[Draw [Histogram] [Mesh] [Monochrome] [Add]]

If the user specifies **draw histogram**, then histograms are drawn in a separate graphics window. The window contains one histogram for each quality metric. If the user specifies draw mesh, then the mesh elements are drawn in the default graphics window. The histogram and mesh graphics are color coded by quality: a small metric value corresponds to red, a large metric value to blue and in-between values according to the rainbow. If monochrome is specified, then the graphics are not color-coded. If add is specified, then the current display is not cleared before drawing the mesh elements.

List

[List [Detail] [Id] [Verbose Errors]]

If the user specifies **List**, then the quality data is summarized in text form. **List Detail** lists the mesh elements by ascending quality metric. **List Id** lists the ids of the mesh elements. If **Verbose Errors** is specified, then details about unacceptable quality elements are printed out above the summaries.

Filter

There are several options available to *filter* the output of the quality command, using the following filter options :

[High <value>] [Low <value>]

Discards elements with metric values above or below value; either or both can be used to get elements above or below a specified value or in a specified range.

[Top <number>] [Bottom <number>]

Keeps only number elements with the highest or lowest metric values. For example, " **Quality hex all top 10** " would request the elements with the 10 highest values of the aspect ratio metric.

Mesh Quality Example Output

The typical summary output from the command **quality surface 24** is shown in Figure 1. Figure 2 shows the corresponding histogram. The colored element display resulting from the **command quality surface 1 draw `Skew'** is shown Figure 3. A color legend is also printed to the console as shown in Figure 4.

```
Surface 24 Quad quality, 292 elements:
--------------------------------------
  Function Name   Average    Std Dev     Minimum   (id)    Maximum  (id)
----------------  ---------  ---------   --------------    -------------
   Aspect Ratio 1.339e+00  3.374e-01   1.001e+00 (244)  3.662e+00 (132)
           Skew 1.848e-01  1.461e-01   7.986e-04 (212)  6.440e-01 (284)
          Taper 1.342e-01  9.397e-02   8.689e-03 (164)  5.500e-01 (133)
        Warpage 9.991e-01  4.465e-03   9.283e-01 (14)  1.000e+00 (82)
   Element Area 6.075e-04  4.725e-04   4.941e-05 (248)  2.202e-03 (274)
        Stretch 7.276e-01  1.233e-01   3.266e-01 (147)  9.587e-01 (161)
  Maximum Angle 1.099e+02  1.329e+01   9.079e+01 (82)  1.738e+02 (14)
  Minimum Angle 7.143e+01  1.185e+01   3.373e+01 (135)  8.955e+01 (82)
  Condition No. 1.250e+00  6.244e-01   1.003e+00 (161)  1.107e+01 (14)
       Jacobian 5.125e-04  4.273e-04   9.696e-06 (14)  1.918e-03 (274)
Scaled Jacobian 9.044e-01  1.104e-01   1.072e-01 (14)  9.999e-01 (82)
          Shear 9.045e-01  1.104e-01   1.072e-01 (14)  9.999e-01 (82)
          Shape 8.436e-01  1.314e-01   9.033e-02 (14)  9.966e-01 (161)
  Relative Size 3.036e-01  2.531e-01   3.226e-03 (248)  9.710e-01 (45)
 Shear And Size 2.789e-01  2.361e-01   1.477e-03 (14)  9.389e-01 (45)
 Shape And Size 2.609e-01  2.234e-01   1.245e-03 (14)  9.389e-01 (45)
     Distortion 8.118e-01  1.352e-01   9.654e-02 (14)  9.864e-01 (82)
```

Figure 1. Typical Summary for a Quality Command

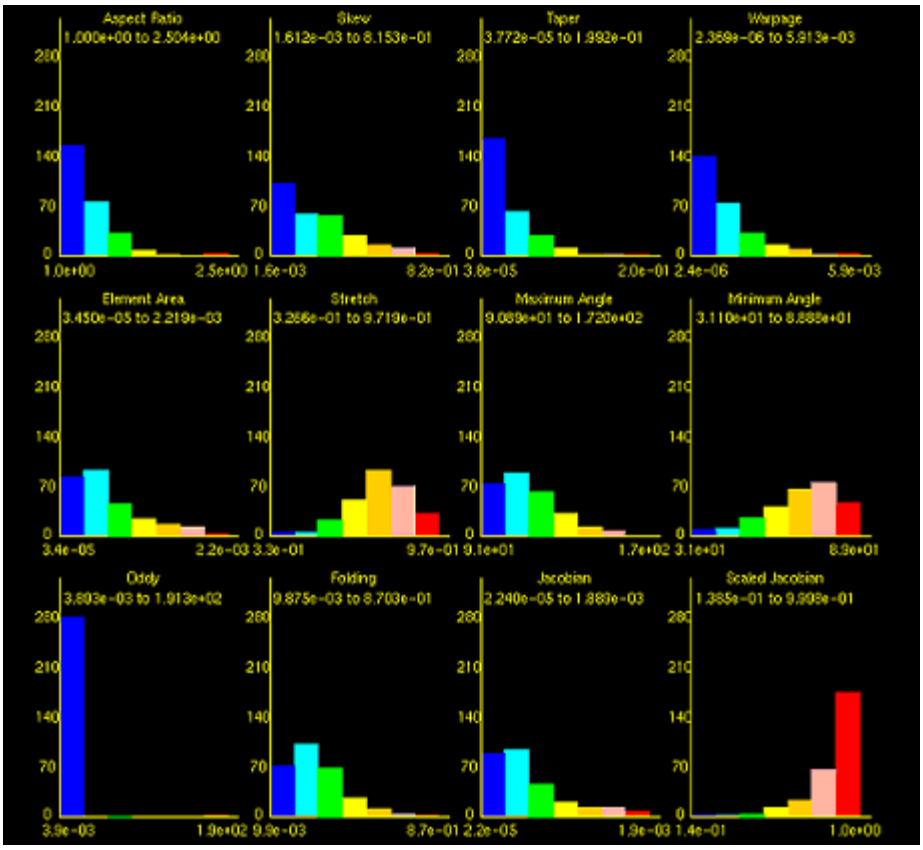


Figure 2. Histogram output from command "Quality Surface 24 Draw Histogram"

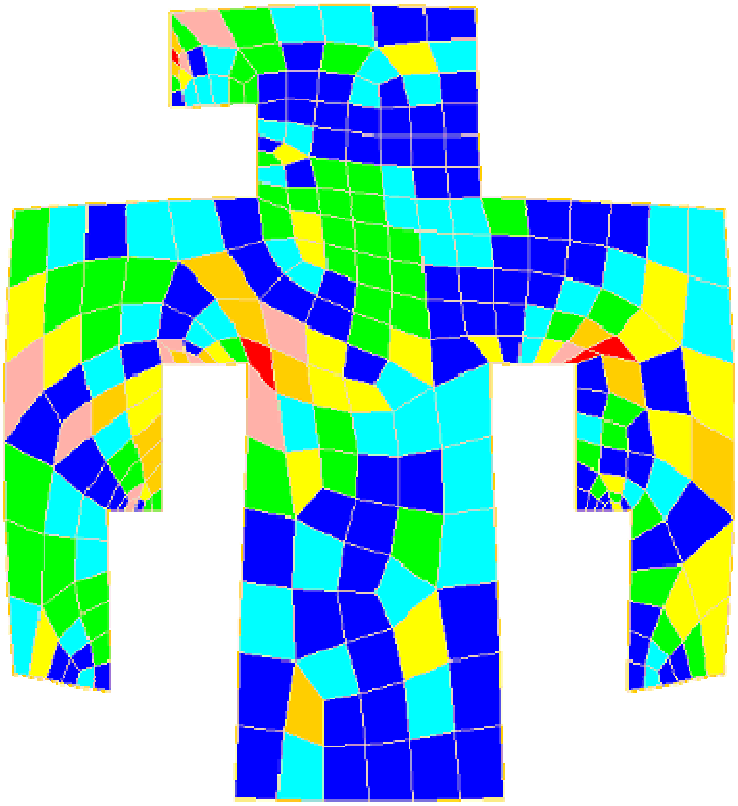


Figure 3. Graphical output of quality metric for command "Quality Surface 24 Skew Draw Mesh"

```
Surface 24 Quad quality, 280 elements:
    Skew ranges from 1.612e-03 to 8.153e-01 (280 entities)
    Blue ranges from 1.612e-03 to 1.178e-01 (102 entities)
    Cyan ranges from 1.178e-01 to 2.341e-01 (60 entities)
   Green ranges from 2.341e-01 to 3.503e-01 (58 entities)
  Yellow ranges from 3.503e-01 to 4.666e-01 (29 entities)
DkYellow ranges from 4.666e-01 to 5.828e-01 (15 entities)
    Pink ranges from 5.828e-01 to 6.990e-01 (12 entities)
     Red ranges from 6.990e-01 to 8.153e-01 (4 entities)
```

Figure 4. Legend for command "Quality Surface 1 Skew Draw Mesh"

Automatic Mesh Quality Assessment

CUBIT performs an automatic calculation of mesh quality which warns users when a particular meshing scheme has created a mesh whose quality may be inadequate. These warnings are supplied in case the user forgets to manually check the mesh quality.

The SHEAR quality of the mesh is automatically computed as part of the command "Mesh <entity_id_list>" (e.g., "Mesh Volume 1). The shear quality for the mesh is the minimum value of the shear metric over all the elements. The "shear" metric measures element skew and ranges between zero and one, with a value of zero signifying a non-convex element, and a value of one being a "perfect" element.

If the minimum shear quality of the mesh created is zero, the code reports "ERROR: Negative Jacobian Element Generated" to the command window. If the minimum shear quality of the mesh is positive but less than a certain threshold, the code reports "WARNING: Poorly-Shaped Element Generated" to the command window. Also reported in this case is the ID of the offending element, the value of its shear metric, and the value of the threshold to which it was compared. The default value of the threshold parameter is 0.2. Users may change the threshold value by issuing the command

[set] shape threshold <double=0.2>

Controlling Mesh Quality

If the quality of a model after meshing isn't acceptable, there are two options available to improve that quality. The user can ask for more smoothing, or delete the mesh and start over. There are some commands that the user can invoke before meshing the model which can help to improve mesh quality. Some of them are discussed here.

Skew Control

The philosophy behind the skew control algorithm is one of subdividing surfaces into blocky, four-sided areas which can be easily mapped. The goal of this subdivide-and-conquer routine is to lessen the skew that a mesh exhibits on submapped regions. By controlling the skew on these surfaces, the mesh of the underlying volume will also demonstrate less skew.

The commands for skew control are:

Control Skew Surface <surface_id_range> [Individual]

Delete Skew Control Surface {surface_list} [Propagate]

The keyword **Individual** is deprecated. It's purpose is to specify that surfaces should be processed without regards to the other surfaces in the given list. This is not necessary, and could lead to problems with the final mesh. When the command is entered, the algorithm immediately processes the surfaces, inserting vertices and setting interval constraints on the resulting subdivided curves. In this way the mesh is more constrained in its generation, and the resulting skew on the model can be lessened. The only surfaces which can utilize this algorithm are those which lend themselves to a structured meshing scheme, although future releases might lessen this restriction.

The user also has the ability to delete the changes that the skew control algorithm has made. This is done by using the **delete skew control** command.

When the user requests the deletion of the skew control changes on a given surface, every curve on that surface will have the skew control changes deleted, even if a given curve is shared with another surface on which skew control was performed. If the user wishes to propagate the deletion of skew control to all surfaces which are affected by one (or more) particular surfaces, the keyword **propagate** should be used.

Propagate Curve Bias

When a bias mesh scheme is applied to a curve, this sometimes creates skewing of the surface mesh which is attached. Sometimes the you will want to ensure that the same bias is applied to curves on attached surfaces so that this skewing is minimized. The command for doing this is:

Propagate Curve Bias [surface|volume|body|group <id_list>]

This command will search out all 4-sided mapable surfaces in the input list, find which curves of those have a bias scheme set, and will propagate that bias across the mapable surfaces.

Adjust Boundary

Adjust Boundary {Surface|Group} <id_range> [Angle <double>]

This command can be used to improve element quality for mapped or submapped surface meshes. Often, due to vertex positions, the curve meshing for a surface will lead to a poor quality surface mesh. This command can be used to adjust the curve meshes in an attempt to generate a better quality surface mesh. The command works by looking at the angle the mesh edges leave the boundary. In a perfect mapped or submapped mesh, the mesh edges will be orthogonal to the boundary, or will go off at 90 degree angles. The adjust boundary command looks at the deviation of the mesh edges, and if it is greater than the prescribed angle deviation, it will move the node location such that it is 90 degrees, if possible. The deviation angle by default is 5 degrees and can be changed by the user through the **[Angle <double>]** option in the command. In order to modify the curve meshes, the surface meshes are first deleted then later remeshed after the curve meshes have been repositioned and fixed. This command assumes that the volumes attached to the surface have not be meshed, if they have been, the command will return an error message. It should be noted that this command, while useful, may not always work due to interval constraints (i.e., you may need to change the intervals on the surface), or if the surfaces are not very blocky.

Coincident Node Check

The ability to check for coincident nodes in the model is available in CUBIT. It uses an efficient octal hash tree to make the comparisons. The command is:

> Quality Check Coincident Node [in] [group | body | volume | surface | curve | vertex <id_range>] [merge [delete]] [HIGHLIGHT | draw [color <number>]] [list] [into group [name | id]]

If no entity list is given, the command works on all the nodes in the model. If an entity list is given, than it compares the nodes on those entities *with the rest of the nodes in the model*. By default the command highlights the coincident nodes in the graphics and lists the total number of coincident nodes found. You can also have it clear the graphics and draw the nodes, and/or list the coincident node ids. Optionally, the coincident nodes found can be placed in a group.

If the model being operated on is from an imported universal file (i.e., no geometry exists in the model), you can merge the coincident nodes with the *merge* option. In this case *delete* allows you to delete the extra nodes (recommended). If you do not delete them they are placed into an output group.

You can control the tolerance used to check between nodes with the following setting (default = 1e-8):

> set Node Coincidence Tolerance [<value>]

Mesh Modification

- Mesh Smoothing
- Mesh Refinement
- Mesh Coarsening
- Mesh Grafting
- Mesh Cutting
- Node and Nodeset Repositioning
- Collapsing Mesh Edges

After meshing is completed, it may be desirable to change features of the mesh without remeshing the whole volume. Mesh modification methods include tools for improving mesh quality, repositioning mesh elements, grafting separate meshes together, or changing mesh density. These methods can be applied on the whole model, or on small sections of the model without requiring remeshing the geometry, and without modifying the underlying geometry.

Mesh Smoothing

- Centroid Area Pull
- Equipotential
- Laplacian

- Condition Number
- Mean Ratio
- Winslow

Related Topics

- Smoothing facet-based surfaces

After generating the mesh, it is sometimes necessary to modify that mesh, either by changing the positions of the nodes or by removing the mesh altogether. CUBIT contains a variety of mesh smoothing algorithms for this purpose. Node positions can also be fixed, either by specific node or by geometry entity, to restrict the application of smoothing to non-fixed nodes.

Mesh smoothing in CUBIT operates in a similar fashion to mesh generation, i.e. it is a two-step process whereby a smooth scheme is chosen and set, then a smooth command performs the actual smoothing. Like meshing algorithms, there is a variety of smoothing algorithms available, some of which apply to multiple geometry entity types and some which only apply to one specific type (these algorithms are described below.) To smooth the mesh on a geometry entity, the user must perform the following steps:

1. Set the smooth scheme for the object using the following command:

   {Curve|Surface|Volume} <range> smooth scheme <scheme>

   where **<scheme>** is any acceptable smooth scheme described in this section. Also set any scheme-specific information, using the smooth scheme setting commands described below.

2. Smooth the object, using the command:

   Smooth {Curve|Surface|Volume|Hex|Tet} <range>

Groups of entities may be smoothed, by smoothing a group or a body.

If a Body is specified, the volumes in that Body are smoothed. If a Group is specified, only the volume meshes within these groups are smoothed - no smoothing of the surface meshes is performed.

Typically, smoothing algorithms move nodes in order to improve the quality of the mesh on a given geometry entity. Smoothing is terminated either by satisfying a smoothing tolerance or by performing the maximum number of smoothing iterations. The smooth tolerance may be set by the user:

[Set] Smooth Tolerance <tol>

The value <tol> tells the smoother to stop when node movement is less than tol * the local minimum edge length. The default value for tol is 0.05. The maximum number of iterations may be set by the user: For volumes, the smooth tolerance and iterations may be set by the user but they are presently ignored by the smoothers:

Volume Smooth Tolerance <tol>

Volume Smooth Iterations <iters>

The default values are 0.05 for the tolerance and 18 * (number of hexes / number of nodes)^(1/3)

Where used in the smooth schemes below, the Free keyword permits the nodes lying on the bounding entities to "float" along those entities; without this keyword, boundary nodes remain fixed.

Nodal positions may be fixed so that no smoothing scheme, either implicit or explicit, will move them, with the following command:

{Curve|Surface|Volume} <range> Node Position {Fixed|Free}

Node <range> Position {Fixed|Free}

The following command does not fix nodal positions, but does fix the connectivity of the mesh, preventing certain volume schemes from changing the bounding mesh:

{Curve|Surface|Volume} Mesh {Fixed|Free}

The specific smooth schemes available in CUBIT are now described in detail.

The additional following schemes are currently being researched (i.e. use at your own risk!):

- Randomize
- Untangle

Centroid Area Pull

Applies to: Surface Meshes

Summary: Attempts to create elements of equal area

Syntax:

Surface <range> Smooth Scheme Centroid Area Pull [Free]

Discussion:

This smooth scheme attempts to create elements of equal area. Each node is pulled toward the centroids of adjacent elements by forces proportional to the respective element areas (Jones, 74).

Equipotential

Applies to: Volume Meshes

Summary: Attempts to equalize the volume of elements attached to each node

Syntax:

Volume <range> Smooth Scheme Equipotential [Free]

Discussion:

This smoother is a variation of the Equipotential (Jones, 74) algorithm that has been extended to manage non-regular grids (Tipton, 90). This method tends to equalize element volumes as it adjusts nodal locations. The advantage of the equipotential method is its tendency to "pull in" badly shaped meshes. This capability is not without cost: the equipotential method may take longer to converge or may be divergent. To impose an equipotential smooth on a volume, each element must be smoothed in every iteration--a typically expensive computation. While a Laplacian method can complete smoothing operations with only local nodal calculations, the equipotential method requires complete domain information to operate.

Laplacian

Applies to: Curve, Surface, and Volume meshes

Summary: Tries to make equal edge lengths

Syntax:

{Surface|Volume} <range> Smooth Scheme Laplacian [Free] [Global]

Discussion:

The length-weighted Laplacian smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement (Jones, 74). Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions.

Currently, the stopping criterion for curve smoothing is 0.005, i.e., nodes are no longer moved when smoothing moves the node less than 0.005 * the minimum edge length. The maximum number of smoothing iterations is the maximum of 100 and the number of nodes in the curve mesh. Neither of these parameters can currently be set by the user.

Using the **global** keyword when smoothing a group of surfaces will allow smoothing of mesh on shared curves to improve the quality of elements on both surfaces sharing that curve.

Condition Number

Applies to: Triangular or Quadrilateral Surface Meshes, Tetrahedral or Hexahedral Volume Meshes. Does not apply to Mixed Element Meshes such as those resulting from scheme HexTet.

Summary: Optimizes the mesh condition number to produce well-shaped elements.

Syntax:

> Surface <surface_id_range> Smooth Scheme Condition Number [beta <double=2.0>] [cpu <double=10>]

Related Commands:

> Untangle

Discussion:

The condition number smoother is designed to be the most robust smoother in Cubit because it guarantees that if the initial mesh is non-inverted then the smoothed mesh will also be non-inverted. The price exacted for this capability is that this smoother is not as fast as some of the other smoothers.

Condition Number measures the distance of an element from the set of degenerate (non-convex or inverted) elements. Optimization of the condition number increases this distance and improves the shape quality of the elements. Condition number optimization requires that the given mesh contain no negative Jacobians. If the mesh contains negative Jacobians and this command is issued, Cubit automatically calls the Untangle smoother and attempts to remove the negative Jacobians. If successful, condition number smoothing occurs next; the resulting mesh should have no negative Jacobians. If untangling is unsuccessful, condition number smoothing is not performed.

There is no "fixed/free" option with this command; boundary nodes are always held fixed.

The command above only sets the smoothing scheme; to actually smooth the mesh one must subsequently issue the command "smooth surface <surface_id_range>" or "smooth volume <volume_id_range>".

Stopping Criteria: Smoothing will proceed until the objective function has been minimized or until one of two user input stopping criteria are satisfied. To input your own stopping criterion use the optional parameters 'beta'and 'cpu' in the command above. The value of beta is compared at each iteration to the maximum condition number in the mesh. If the maximum condition number is less than the value of beta, the iteration halts. In Cubit condition number ranges from 1.0 to infinity, with 1.0 being a perfectly shaped element. Thus the smaller the maximum condition number, the better the mesh shape quality. The default value of the beta parameter is 2.0. The value supplied for the "cpu" stopping criterion tells the code how many minutes to spend trying to optimize the mesh. The default value is 10 minutes. Optimization may also be halted by using "control-C" on your keyboard.

To view a detailed report of the smoothing in progress issue the command "set debug 91 on" prior to smoothing the surfaces or volumes. You will get a synopsis of whether or not untangling is needed first and whether the stopping criteria have been satisfied. In addition the following printout information is given for each iteration of the conjugate gradient numerical optimization:

> **Iteration=n, Evals=m, Fcn=value1, dfmax=value2, time=value3 ave_cond=value4, max_cond=value5, min_jsc=value6**

n is the iteration count, **m** is the number of objective function evaluations performed per iteration, **value1** is the value of the objective function (this usually decreases monotonically), **value2** is the norm of the gradient (does not always decrease monotonically), and **value3** is the cumulative cpu time (in seconds) spent up to the current iteration. The minimum possible value of the objective function is zero but this is attained only for a perfect mesh. **ave_cond**, **max_cond**, and **min_jsc** are the average and maximum condition number, and the minimum scaled jacobian. **ave_cond** generally decreases monotonically because it is directly related to **value1**.

Mean Ratio

Applies to: Triangular or Quadrilateral Surface Meshes, Tetrahedral or Hexahedral Volume Meshes. Does not apply to Mixed Element Meshes such as those resulting from scheme HexTet.

Summary: Moves interior mesh nodes to optimize the average mean ratio metric value of the mesh.

Syntax:

Surface <surface_id_range> Smooth Scheme Mean Ratio [cpu <double=10>]

Volume <volume_id_range> Smooth Scheme Mean Ratio [cpu <double=10>]

Discussion:

CUBIT includes a mean ratio smoother provided by MESQUITE, a mesh optimization toolkit by Argonne National Laboratory and Sandia National Laboratories. (See Brewer, et al. 2003 for more details on the MESQUITE toolkit.) This smoother is similar in purpose to the Condition Number smoother. However, the Mean Ratio smoother uses a second order optimization method, and therefore it will often reach a near-optimal mesh more quickly than the Condition Number smoother. The Mean Ratio smoother requires the initial mesh to be untangled, but the smoother is guaranteed to not tangle the mesh. If the user attempts to call the Mean Ratio smoother on a tangled mesh, an untangler will first attempt to untangle the mesh before calling the Mean Ratio smoother.

The Mean Ratio smoother's optimization process terminates when one of the following three criteria is met:

1. The mesh is "close" to an optimal mesh configuration.
2. The maximum allotted time has been exceeded.
3. The user interrupts the smoothing process.

The user has control over the second and the third criteria only. For criterion 2, the default is for the smoother to terminate after ten minutes even if a near-optimal mesh has not been reached. The user can change this time bound by specifying the optional "cpu" argument in the command listed above. This argument takes a single, positive number that represents the time (in minutes) that will be used as a time bound. If the user wishes to terminate the process early, criteria three allows the user to "interrupt" (for example, on some platforms, by pressing CTRL-C) the process. If the process is terminated early, the mesh will not revert to the original node positions; CUBIT will instead keep the partially optimized mesh.

Winslow

Applies to: Surface meshes

Summary: Elliptic smoothing technique for structured and unstructured surface meshes

Syntax:

Surface <range> Smooth Scheme Winslow [Free]

Discussion:

Winslow elliptic smoothing (Knupp, 98) is based on solving Laplaces equation with the independent and dependent variables interchanged. The method is widely used in conjunction with the mapping and submapping methods to give smooth meshes with positive Jacobians, even on non-convex two-dimensional regions. The method has been extended in CUBIT to work on unstructured meshes.

Mesh Refinement

- Uniform Mesh Refinement
- Refining at a Geometric or Mesh Feature
- Hexahedral Refinement Using Sheet Insertion

CUBIT provides several methods for *conformally* refining an existing mesh. Conformal mesh refinement does not leave hanging nodes in the mesh after refinement operations, rather conformal mesh refinement provides transition elements to the existing mesh. Both local and global mesh refinement operations are provided.

Uniform Mesh Refinement

The Refine Surface and Refine Volume commands provide capability for uniformly refining an entire surface or volume mesh. The Refine Surface Command can only be used on surface meshes that are not attached to a volume, whereas the Refine Volume command will refine both surface and volume meshes. The command syntax is:

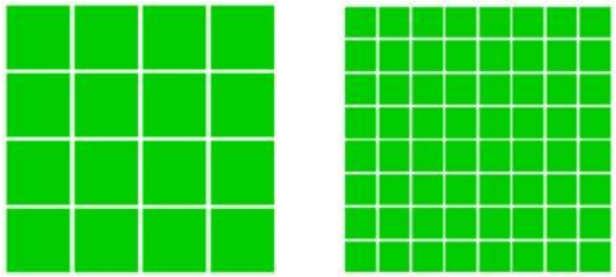
Refine Volume <range>numsplit<int>

Refine Surface <range>numsplit<int>

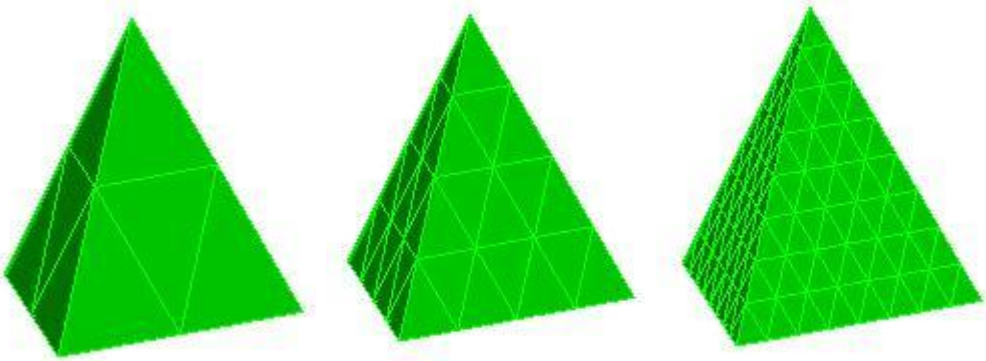
The numsplit option specifies how many times to subdivide an element. A value of 1 will split every triangle and quadrilateral into four pieces, and every tetrahedron and hexahedron into eight pieces. Examples of uniform refinement on each element are shown below. For more information on uniform hexahedral mesh refinement see the documentation for dicing.

original mesh NumSplit = 1 NumSplit = 2

original mesh



NumSplit = 1 NumSplit = 2



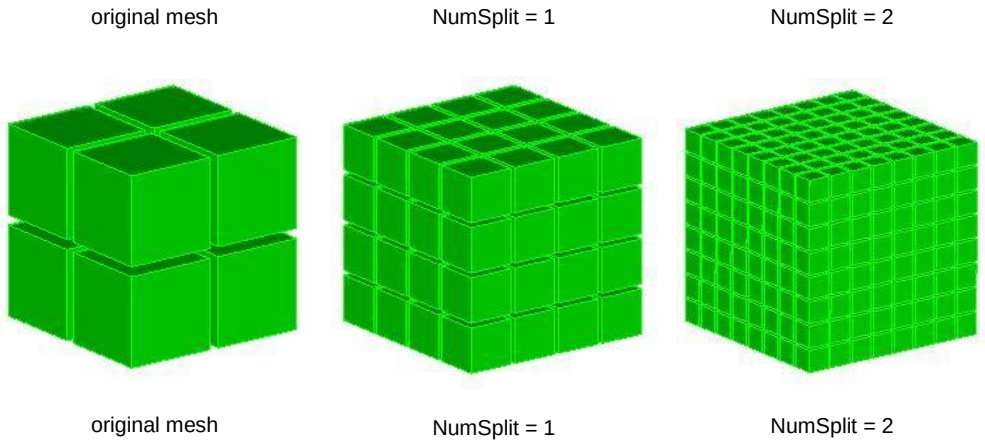


Figure 1. Example of uniform refinement for each of the mesh entities

Refining at a Geometric or Mesh Feature

CUBIT also provides methods for local refinement around geometric or mesh features. Individual elements or groups of elements can be refined in this manner using the following syntax.

Refine {Node|Edge|Tri|Face|Tet|Hex} <range>
[NumSplit<int = 1>|Size <double> [Bias <double>]]
[Depth <int>|Radius <double>] [Sizing_Function]
[no_smooth]

Refine {Vertex|Curve|Surface} <range>
[NumSplit<int = 1>|Size <double> [Bias <double>]]
[Depth <int>|Radius <double>] [Sizing_Function]
[no_smooth]

To use these commands, first select mesh or geometric entities at which you would like to perform refinement. Refinement will be applied to all mesh entities associated with or within proximity of the entities. The all keyword may be used to uniformly refine all elements in the model

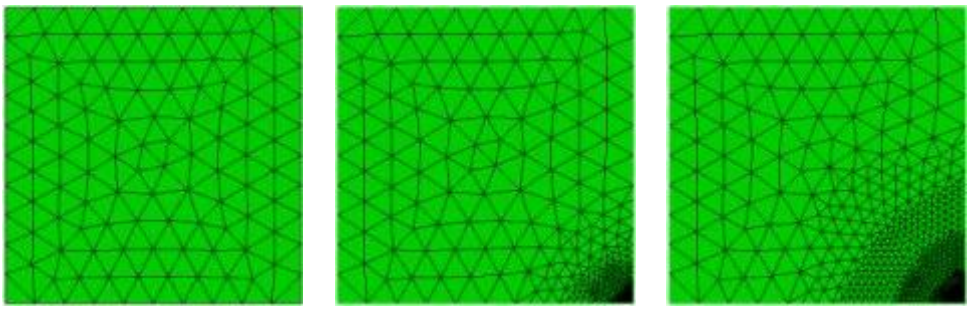
The following is a description of refinement options.

NumSplit

Defines the number of times the elements in the region will be split. A NumSplit value of 1 will split triangles and quadrilaterals into four elements and tetrahedrons and hexahedrons into eight elements. Figure 1 shows each of the geometric entities with a NumSplit value of 0, 1 and 2. The default NumSplit value is 1. Using the refine command without the NumSplit option will split all affected elements once.

Size, Bias

The Size and Bias options are useful when a specific element size is desired at a known location. This might be used for locally refining around a vertex or curve. The Bias argument can be used with the Size option to define the rate at which the element sizes will change to meet the existing element sizes on the model. Figure 2 shows an example of using the Size and Bias options around a vertex. Valid input values for Bias are greater than 1.0 and represent the maximum change in element size from one element to the next. Since refinement is a discrete operation, the Size and Bias options can only approximate the desired input values. This may cause apparent discontinuities in the element sizes. Using the default smooth option can lessen this effect. It should also be noted that the Size option is exclusive of the NumSplit option. Either NumSplit or Size can be specified, but not both.

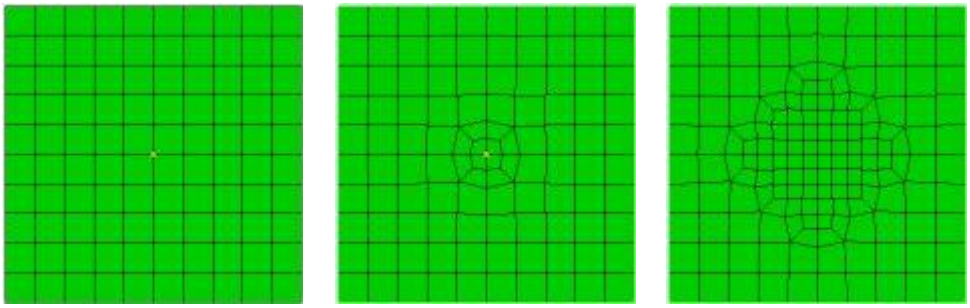


original mesh Bias=2.0 Bias=1.5

Figure 2. Example of using the Size and Bias options at a Vertex.

Depth

The Depth option permits the user to specify how many elements away from the specified entity will also be refined. Default Depth is 1. Figure 3 shows an example of using the depth option when refining at a node.



original mesh Depth=1 Depth=2

Figure 3. Example of using the Depth option at a node to control how far from the node to propagate the refinement.

Radius

Instead of specifying the number of elements to describe how far to propagate the refinement, a real Radius may be entered. The effects of the Radius are similar to that shown in Figure 3, except that the elements whose centroid fall within the specified Radius will be refined. Transition elements are inserted outside of this region to transition to the existing elements.

Sizing Function

Refinement may also be controlled by a sizing function. CUBIT uses sizing functions to control the local density of a mesh. Various options for setting up a sizing function are provided, including importing scalar field data from an exodus file. In order to use this option, a sizing function must first be specified on the surface or volume on which the refinement will be applied. See Adaptive Meshing for a description of how to define a sizing function.

no_smooth

The default mode for refinement operations is to perform smoothing after splitting the elements. This will generally provide better quality elements. In some cases it may be necessary to retain the original node locations after refinement. The no_smooth option provides this capability

Hexahedral Refinement Using Sheet Insertion

Several tools for refining a hexahedral mesh using sheet insertion and deletion are available in CUBIT.

- Refining at a Geometric Feature

- Refining at a Boundary Surface
- Refining along a Path
- Refining a Hex Sheet
- Hex Sheet Drawing

Refining at a Geometric Feature

In addition to uniform refinement, the dicing scheme provides additional controls for specifying refinement options on an existing hex mesh. The following commands offer additional controls on refinement with respect to one or more geometric features of the model.

An existing hexahedral mesh can be refined at a geometric feature using the following command:

Refine Mesh Volume <id> Feature {Surface | Curve | Vertex | Node} <id_range> Interval <integer>

This command refines the mesh around a given feature by adding sheets of hexes. These sheets can be generalized as planes for surfaces, cylinders for curves, and spheres for vertices. The **interval** keyword specifies the number of intervals away from the feature to insert the new sheet of hexes. For this command a single sheet of hexes is inserted into the hexahedral mesh.

Figure 4 shows an example of this command where the feature at which refinement is to be performed is a curve. In this case the interval chosen was, 2. This indicated that the elements 2 intervals away from the curve would be refined.

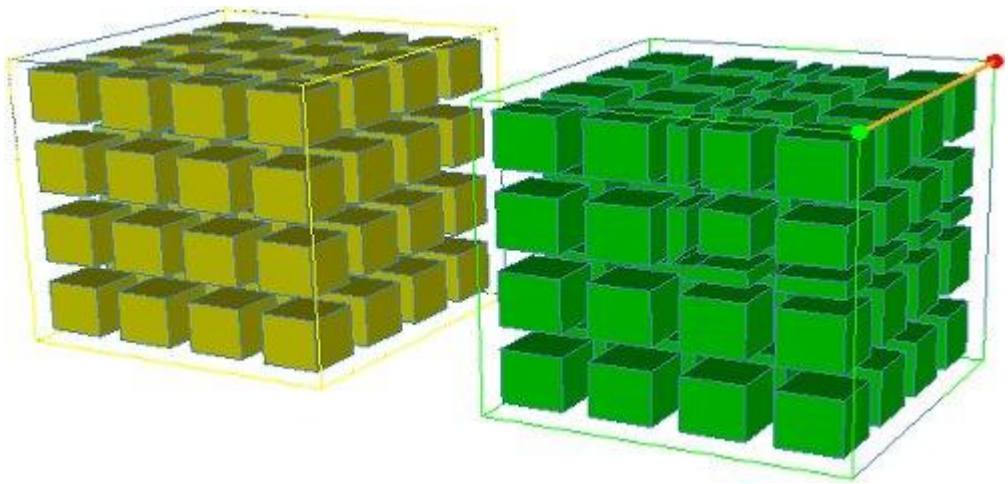


Figure 4. Example of Refinement at a curve

Refinement at a Boundary Surface

Boundary effects to be modeled in the analysis code frequently require a refined mesh near a specific surface. CUBIT provides this capability with the Refine Mesh Boundary command. This command is similar to the Refine Mesh Volume Feature command except that it can insert multiple sheets of hexes near the specified surface.

Refine Mesh Boundary Surface <range> Volume <id> [bias <factor=0.0>] [layer <num_layers=1>] [SMOOTH|no_smooth]

With this command **num_layers** of hexes can be inserted at the first interval from the specified surface. A **bias factor** indicating the change in element size can also be specified. The mesh in Figure 5 with bias 1.5 and layer 5. The default smooth option provides the capability to smooth the mesh following the refinement procedure.

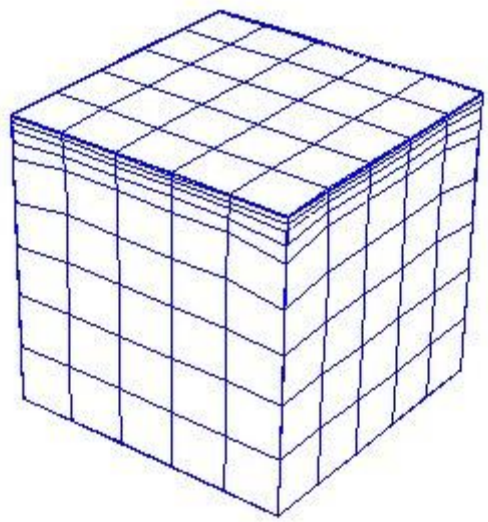


Figure 5. Example of Boundary Surface Refinement

Refining along a path

Hexahedral meshes can be refined from a specific node and along a propagated path using the following command

Refine Mesh Start Node <id> Direction Edge <id> End Node <id> [Smooth]

Figure 6 shows a swept mesh and it's cross section. The cross section view on the left shows a path that has been propagated through the mesh between the start node and end node. This path is then projected along a chain of edges in the direction given by the direction edge as shown in Figure 6 . The start node and end node must be on the same sweep layer. This refinement procedure also requires the volume's meshing scheme to be set to sweep. If the smooth keyword is given the mesh will be smoothed after the refinement step is complete.

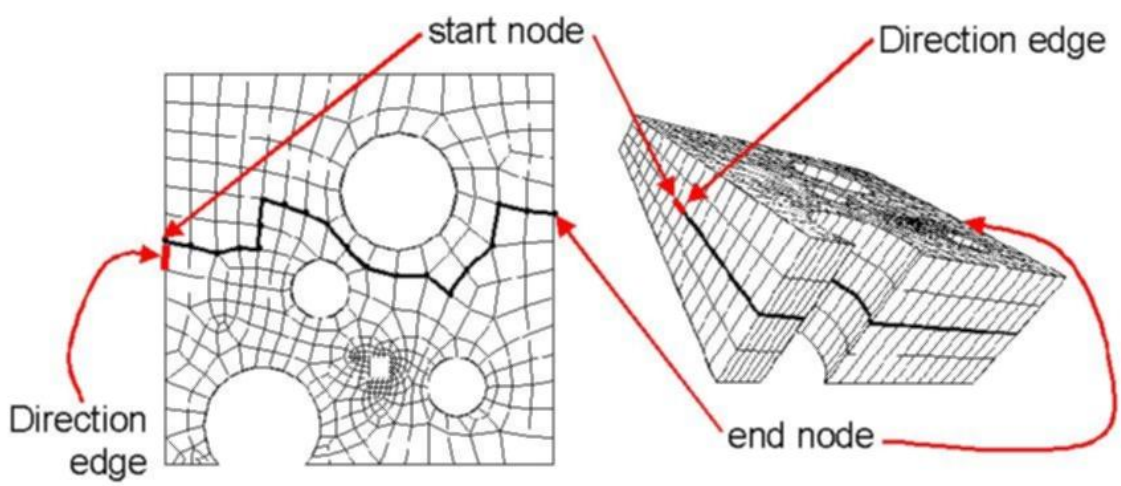


Figure 6. Refining a Mesh Along a Path

Refining a Hex Sheet

Each element in a hex mesh is bisected by three intersecting planes. These planes are referred to as twist planes or whisker sheets. Twist planes can be thought of as general surfaces that propagate through a hex mesh, bisecting each hex element between pairs of faces as shown in Figure 7 . A hex sheet is the collection of hexes that are bisected by a single twist plane.

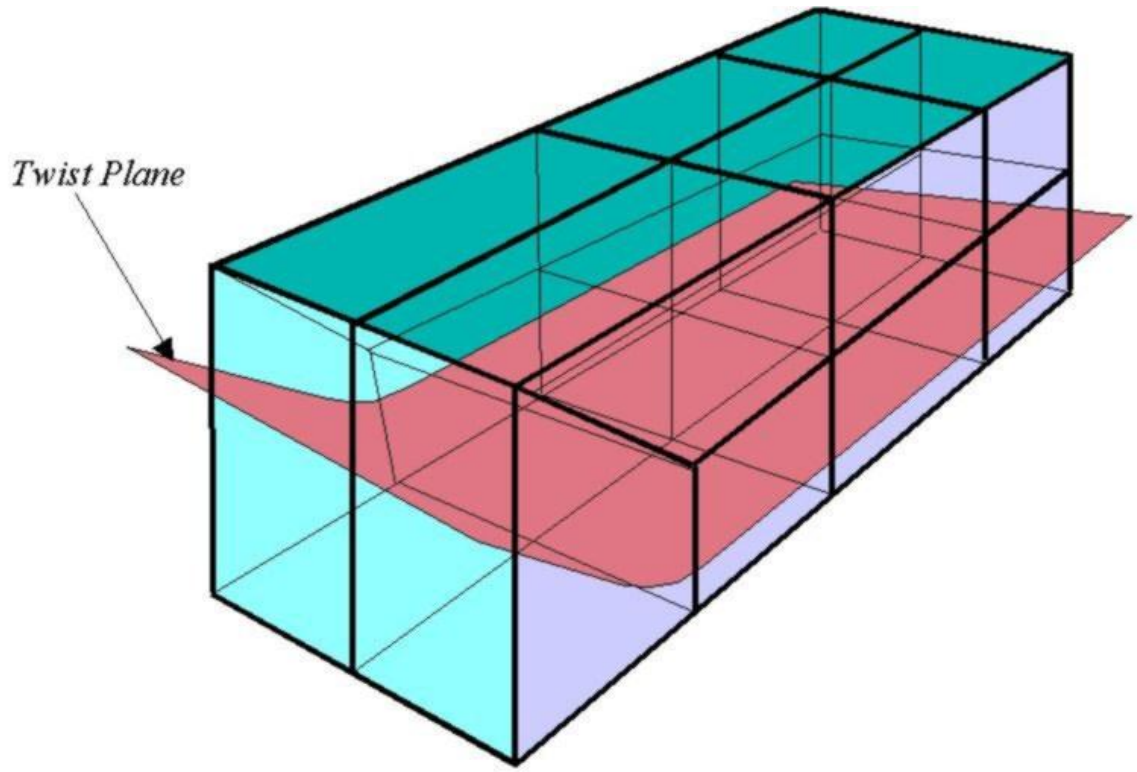


Figure 7. A Twist Plane Bisecting a Sheet of Hex Elements

The following command provide a means to refine a hex mesh at an existing hex sheet.

> Refine Mesh Sheet { Node <id_1> <id_2> | Edge <id_range> } {Factor <double> | greater_than <size>}

The **node** pair or **edge** are used to define the hex sheet. The **factor** and **greater_than** keywords are used to specify the refinement criterion for the sheet. This criterion is based on the distance between opposite faces that are parallel to the twist plane for each hex. If a factor is given, the smallest distance is found in the specified hex sheet, and any hex with a distance greater that the smallest distance multiplied by the factor is refined. If a greater_than value is given, any hex with a distance greater than the given value is refined.

Figure 8 shows an example of this command. In this example, the hex sheet is specified by the edge that is highlighted in the mesh on the left. The factor keyword was used with a value of 2.

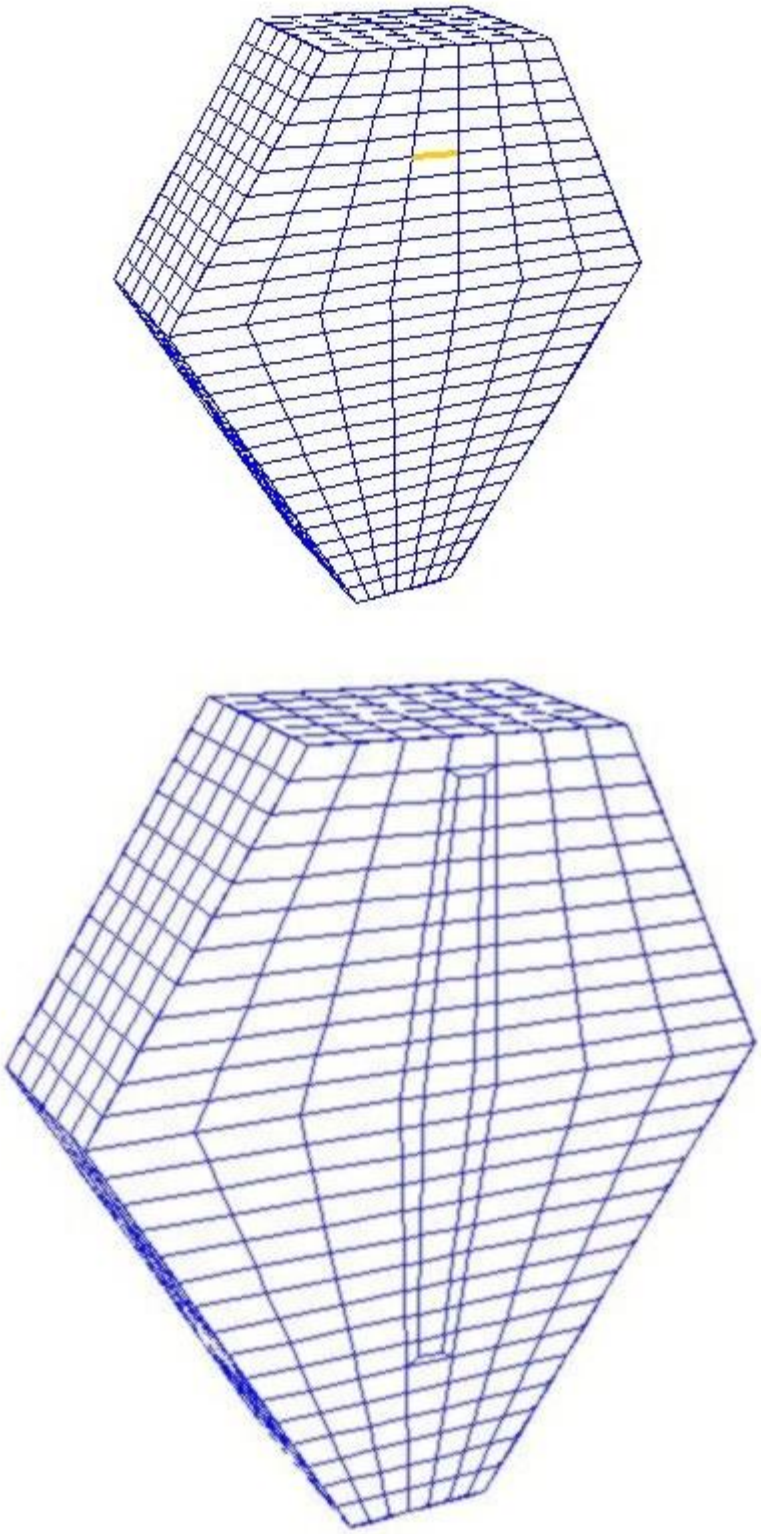


Figure 8. Example of Hex Sheet Refinement

See also Hexahedral Mesh Coarsening for information on how to remove sheets from the hex mesh.

Hex Sheet Drawing

Since refinement of hex meshes generally occurs by inserting hex sheets, tools have been provided to draw a specified sheet or group of sheets.

This command draws a sheet of hexes that is defined by the edge or node pair.

Draw Sheet {edge <id> |node <id_1> <id_2>}[mesh [list]] [color <color_name>] [gradient]

The following command draws the three sheets that intersect to define the given hex. These sheets are drawn green, yellow, and red. To draw a specific sheet, list its color in the command.

Draw Sheet hex <id> [green] [yellow] [red][mesh [list]] [gradient]

The 'gradient' keyword for both commands draws the sheet in gradient shading according to the distance between opposite hex faces that are parallel to the sheet. For the 'draw dicersheet hex ...' command, this option works only if one sheet is being drawn.

The 'mesh' keyword will draw the hexes in the hex sheet. If the 'list' keyword is also give the ids of the hexes in the sheet will be listed.

Mesh Coarsening

- Triangle Coarsening
- Hexahedral Coarsening

Triangle Coarsening

CUBIT provides the capability for coarsening triangle surface meshes. Triangle coarsening uses a technique known as edge collapsing to coarsen a mesh. With this technique, triangle edges are selectively eliminated from the mesh until the specified criteria have been met. The following commands will coarsen an existing triangle surface mesh:

Coarsen {Node|Edge|Tri} <range> {Factor|Size <double> [Bias <double>]} [Depth <int>|Radius <double>] [Sizing_Function] [no_smooth]

Coarsen {Vertex|Curve|Surface} <range> {Factor|Size<double> [Bias<double>]} [Depth<int>|Radius<double>] [Sizing_Function] [no_smooth]

Important: These commands are currently implemented only for *triangle* shaped elements.

To use these commands, first select mesh or geometric entities at which you would like to perform coarsening. Coarsening operations will be applied to all mesh entities associated with or within proximity of the entities. The **all** keyword may be used to uniformly coarsen all triangles in the model.

Following is a description of each of the coarsen options:

Factor

Defines the approximate size relative to the existing edge lengths for which the coarsening will be applied. For example, a factor of 2 will attempt to make every edge length within the specified region approximately twice the size. A factor of 3 will make everything three times the size. Valid input values for factor must be greater than 1. Figure 1 shows an example where a coarsening factor of 2 was applied

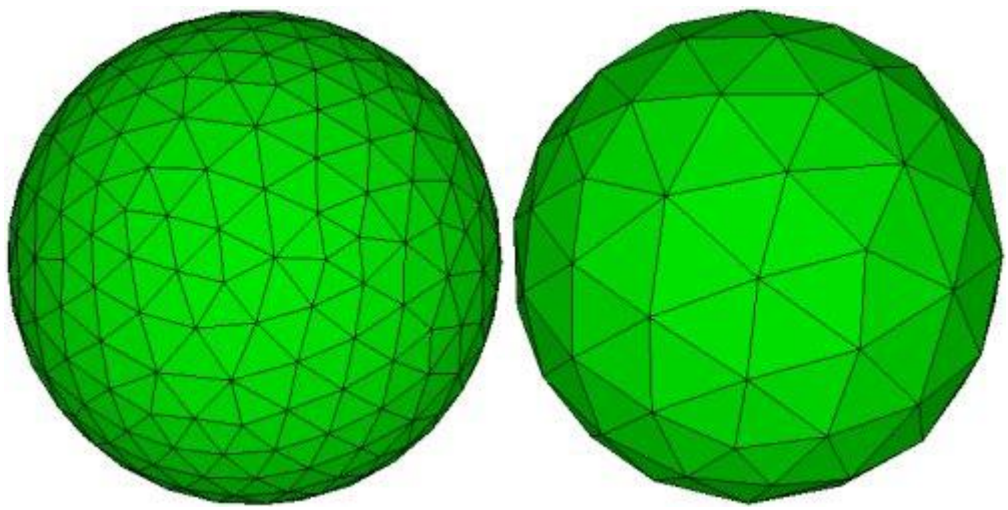


Figure 1. Example of coarsening all triangles with a factor of 2.

Size, Bias

The Size and Bias options are useful when a specific element size is desired at a known location. This might be used for locally coarsening around a vertex or curve. The Bias argument can be used with the Size option to define the rate at which the element sizes will change to meet the existing element sizes on the model. Valid input values for Bias are greater than 1.0 and represent the maximum change in element size from one element to the next. Since coarsening is a discrete operation, the Size and Bias options can only approximate the desired input values. This may cause apparent discontinuities in the element sizes. Using the default smooth option can lessen this effect. It should also be noted that the Size option is exclusive of the Factor option. Either Factor or Size can be specified, but not both.

Depth

The Depth option permits the user to specify how many elements away from the specified entity will also be coarsened. Default Depth is 1.

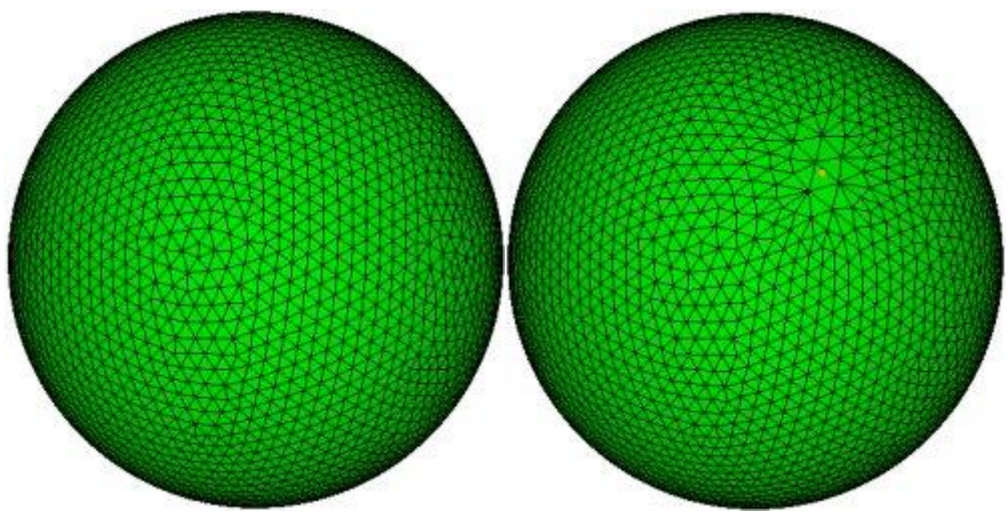


Figure 2. Coarsening performed at a node with factor = 3 and depth = 3

Radius

Instead of specifying the number of elements to describe how far to propagate the coarsening, a real Radius may be entered.

Sizing Function

Coarsening may also be controlled by a sizing function. CUBIT uses sizing functions to control the local density of a mesh. Various options for setting up a sizing function are provided, including importing scalar field data from an exodus file. In order to use this option, a sizing function must first be specified on the surface on which the coarsening will be applied. See Adaptive Meshing for a description of how to define a sizing function.

No_Smooth

The default mode for coarsening operations is to perform smoothing after coarsening the elements. This will generally provide better quality elements. In some cases it may be necessary to retain the original node locations after coarsening. The no_smooth option provides this capability.

Hexahedral Coarsening

CUBIT provides a limited number of options for coarsening hexahedral meshes. The options currently available for hex coarsening rely on the hex sheet extraction process described in Mesh Refinement page. Removing a sheet from a hexahedral mesh essentially means that a complete layer of hexes will be removed and the adjacent layers expanded to take its place.

Extracting a Single Hex Sheet

The following command can be used to extract a single hex sheet.

Extract sheet { Edge <id> | Node <id_1> <id_2> }

The edge or node pair are used to define the sheet that will be extracted. Figure 3 below shows an example of extracting a hex sheet. In this example the hex sheet is specified by the node pair highlighted in the images. Note that the entire layer of hexes between the highlighted nodes has been removed and the neighboring layers have been expanded to take its place.

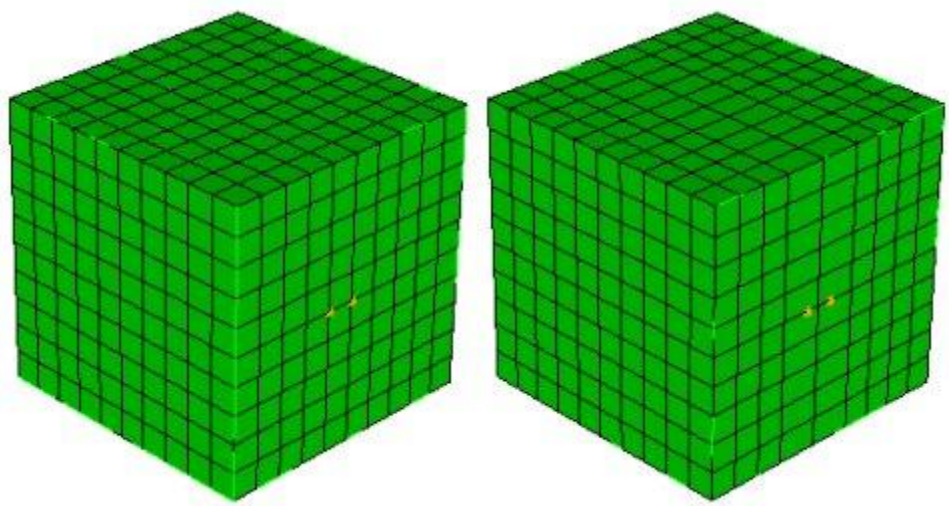


Figure 3. Example of Hex Sheet Extraction

Note: Also see the Mesh Refinement section for a description of hex sheet drawing.

Extracting multiple sheets along a curve

Another option for extracting hex sheets can be done by specifying a curve at which to perform the sheet extraction operations. In this case, multiple layers of hexes can be removed by specifying a curve perpendicular to the hex layers. The command for coarsening perpendicular to a curve is as follows:

Coarsen Mesh Curve <id> Factor <value> [NO_SMOOTH|smooth]

Coarsen Mesh Curve <id> Remove {<num_edges>|edge <id_ranges>} [NO_SMOOTH|smooth]

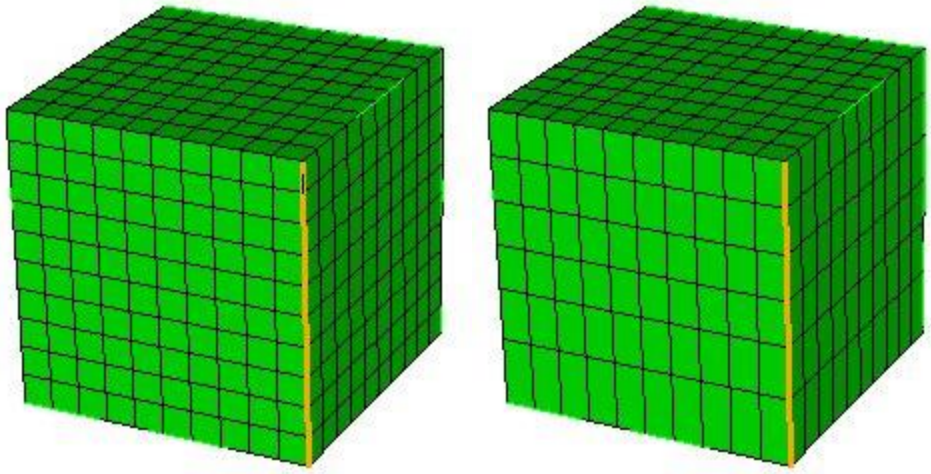


Figure 4. Coarsening a mesh by extracting sheets perpendicular to a curve

The first option uses the **Factor** argument. The factor argument controls how much larger the edges will be on the curve. For example, Figure 4 shows the coarsen mesh curve command used with a factor of 2. In this case, the command attempts to make the mesh edges approximately twice the length relative to their original length along the curve.

The second option uses the **Remove** argument. With this option, a specified number of layers may be removed from the mesh. This may be accomplished by indicating an exact number, or by providing a list of edge IDs that correspond to the layers that will be removed.

The **NO_SMOOTH|smooth** option allows the user to improve the element quality after the sheet extraction process by smoothing the remaining nodes. The default for both of these commands is to not smooth. Smoothing may also be accomplished after sheet extraction by using the smooth volume command.

Uniform hex coarsening

By applying the coarsen mesh curve command multiple times to curves that are orthogonal in the model, the effect of uniform coarsening of the mesh may be achieved.

Mesh Cutting

The term "mesh cutting" refers to modifying an existing mesh by moving nodes to a cutting entity and modifying the connectivity of the mesh so that the original mesh fits a new geometry. The behavior of mesh cutting is intended to be similar to web cutting in that the process results in a decomposition of the original geometry. The difference is that the decomposition is performed on meshed geometry and results in the creation of virtual geometry partitions. The underlying acis body remains unchanged. The user has the option to determine what is partitioned during mesh cutting: the volume, the surfaces only, or nothing.

The current scope of mesh cutting is limited to cutting hex meshed volumes with planes and extended surfaces. These cutting entities are also limited in that mesh cutting will not work if they pass through a vertex at the end of more than two curves. Mesh cutting does not work on tet meshes or surface meshes.

The steps of mesh cutting include:

- **Create a starting mesh**. This mesh is typically simpler than the desired final mesh and can be created with sweeping, mapping, or some other available meshing algorithm. Currently, the starting mesh must be a single volume: mesh cutting does not handle merged volumes or assemblies.
- **Create a cutting entity** that can be used to capture the new detail in the mesh. Currently, mesh cutting works with planes or sheets extended from surfaces. It is important to note that if an extended surface is used, mesh cutting will not capture any geometric features (curves or vertices) of the surface.
- **Issue the command** to cut the mesh. The meshcut commands are similar in syntax and behavior to the webcut commands.

The following entities with the associated commands are available for mesh cutting:

Coordinate Plane

A coordinate plane can be used to cut the model, and can optionally be offset a positive or negative distance from its position at the origin.

Meshcut Volume <range> Plane {xplane|yplane|zplane} [offset <dist>]

The planar surface to be used for mesh cutting can also be previewed using the Draw Plane command.

Planar Surface

An existing planar surface can also be used to cut the model.

Meshcut Volume <range> Plane Surface <surface_id>

The planar surface to be used for mesh cutting can also be previewed using the Draw Plane command.

Plane from 3 points

Any arbitrary planar surface can be used by specifying three nodes that define the plane.

Meshcut Volume <range> Plane Node <3_node_ids>

Extended Surface

An extended surface or "sheet" can also be used for mesh cutting. In this case, the sheet is not restricted to be planar and will be extended in all directions possible. When cutting with an extended surface mesh cutting will ignore all curves and vertices of the surface. Also, the resolution of the mesh will determine how well curved surfaces are captured with meshcutting. A surface with high curvature will not be captured accurately with a coarse mesh. Note that some spline surfaces are limited in extent and may not give an expected result from mesh cutting.

Meshcut Volume <range> Sheet [Extended From] Surface <surface_id>

Note: When cutting with surfaces extended from composite surfaces the default underlying surface approximation may result in a poor final mesh for mesh cutting. This problem can be fixed using the following command:

Composite closest_pt surface <id> gme

See the discussion on composite geometry for a more detailed description of this command.

Meshcut Options

The following options can be used with all the meshcut commands:

[PARTITION VOLUME|partition surface|no_partition]: By default, mesh cutting will create virtual partitions of the volume being cut to match the cutting entity. This option allows mesh cutting to also create only the surface partitions or create no partitions for the volume or surfaces.

[no_refine]: This option tells mesh cutting not to refine the mesh around the cutting entity.

[no_smooth]: This option tells mesh cutting not to perform the final smoothing step after the cut has been made.

Meshcutting Scope

The following is a list of the current scope and limitations of meshcutting.

- Meshcutting only works on hex meshes.
- Meshcutting only works for single volumes. It currently does not handle assembly meshes.
- Currently, only planes and extended surfaces can be used as the cutting entity.
- Curves and vertices on the cutting entity will not be captured in the mesh.

- Meshcutting will not work if the cutting entity passes through a meshed vertex that is at the end of more than two curves.
- The resolution of the mesh determines how well a non-planar cutting entity will be captured in the resulting mesh. Small features and high curvature will not be captured by a coarse mesh.
- Spline surfaces are limited in extent and may not give expected results if used as an extended cutting surface.

Meshcutting Example

The figures below show an example of mesh cutting. Figure 1 shows the body that will be meshed. This body is a brick with intersecting through-holes. The steps to create a mesh for this body are listed below.

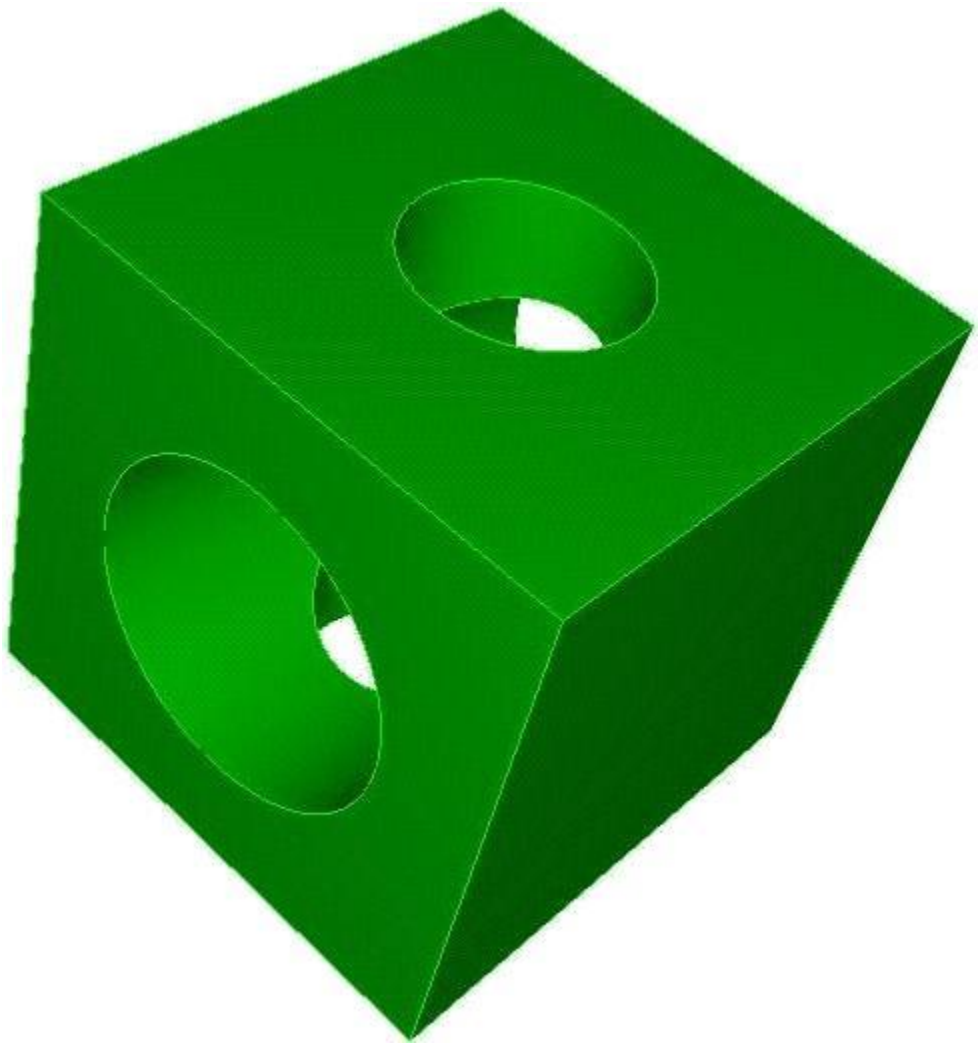


Figure 1: The original, unmeshed body

Step 1: Create a starting mesh. Figure 2 below shows the starting mesh for this problem. The commands for this mesh are:

```
cubit> create brick x 10
cubit> create cylinder radius 3 z 15
cubit> subtract 2 from 1
cubit> volume 1 scheme sweep
cubit> volume 1 size .75
cubit> mesh volume 1
```

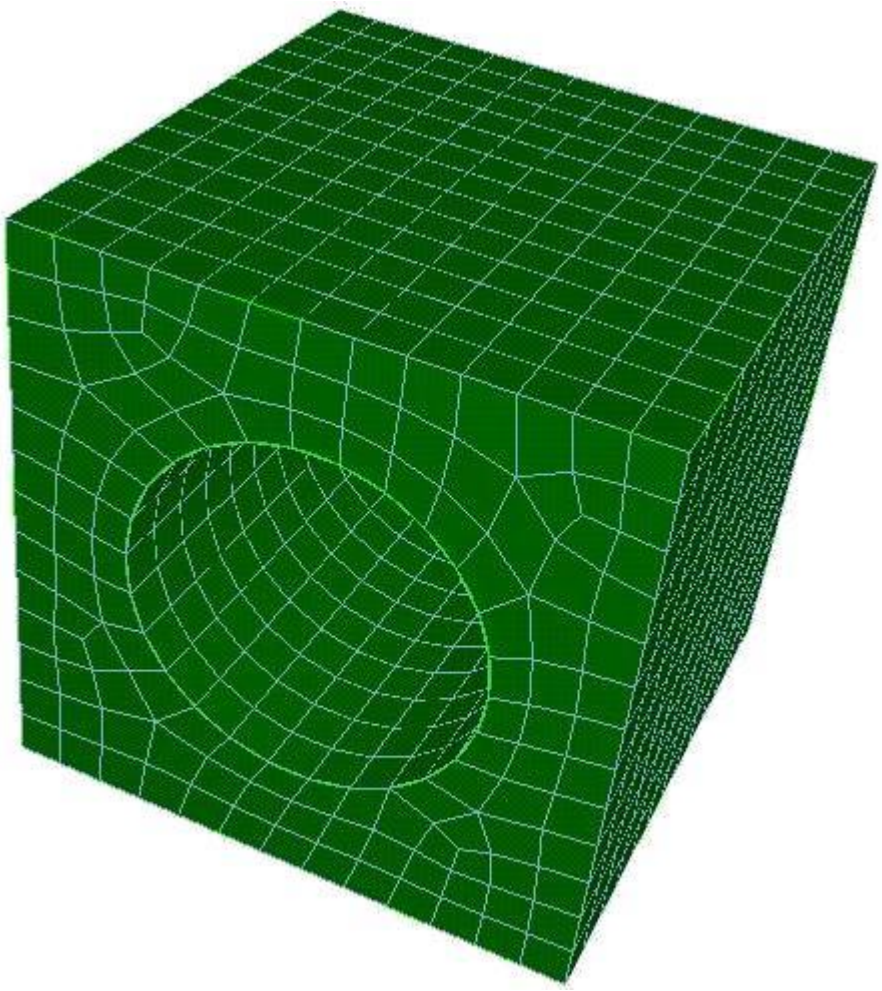


Figure 2: The starting mesh

Step 2: Create a cutting entity. Figure 3 shows the volume that will be used to cut the mesh. The commands are:

```
cubit> create cylinder radius 2 z 15
cubit> rotate body 3 about x angle 90
```

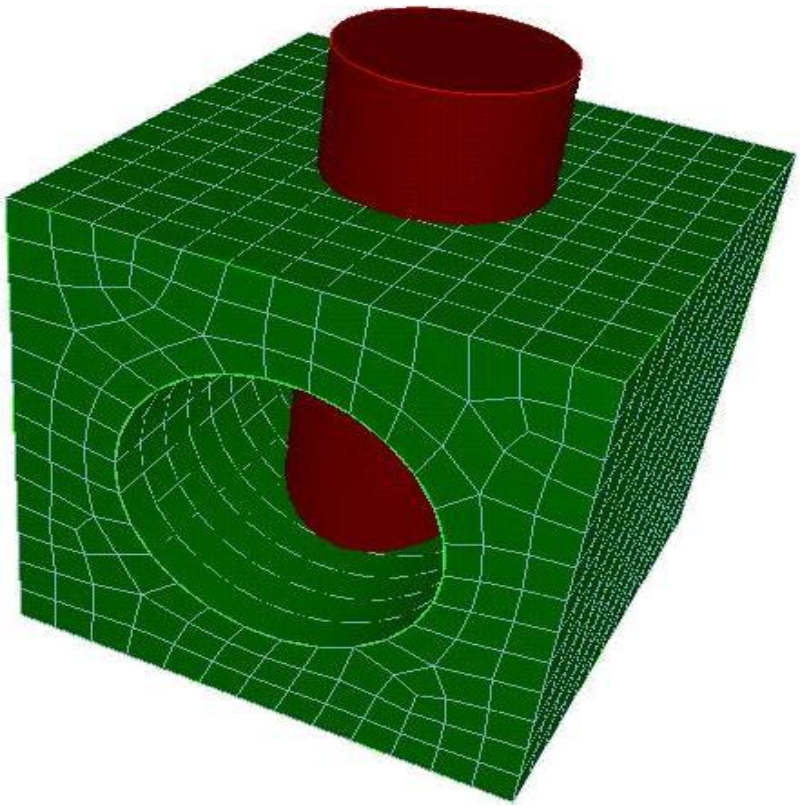


Figure 3: The starting mesh and cutting entity

Step 3: Cut the mesh. Figure 4 shows the new mesh after the original mesh has been cut. At this point we have 3 meshed volumes. The commands for this step are:

```
cubit> meshcut vol 1 sheet surface 13
cubit> draw volume 4 5 6
```

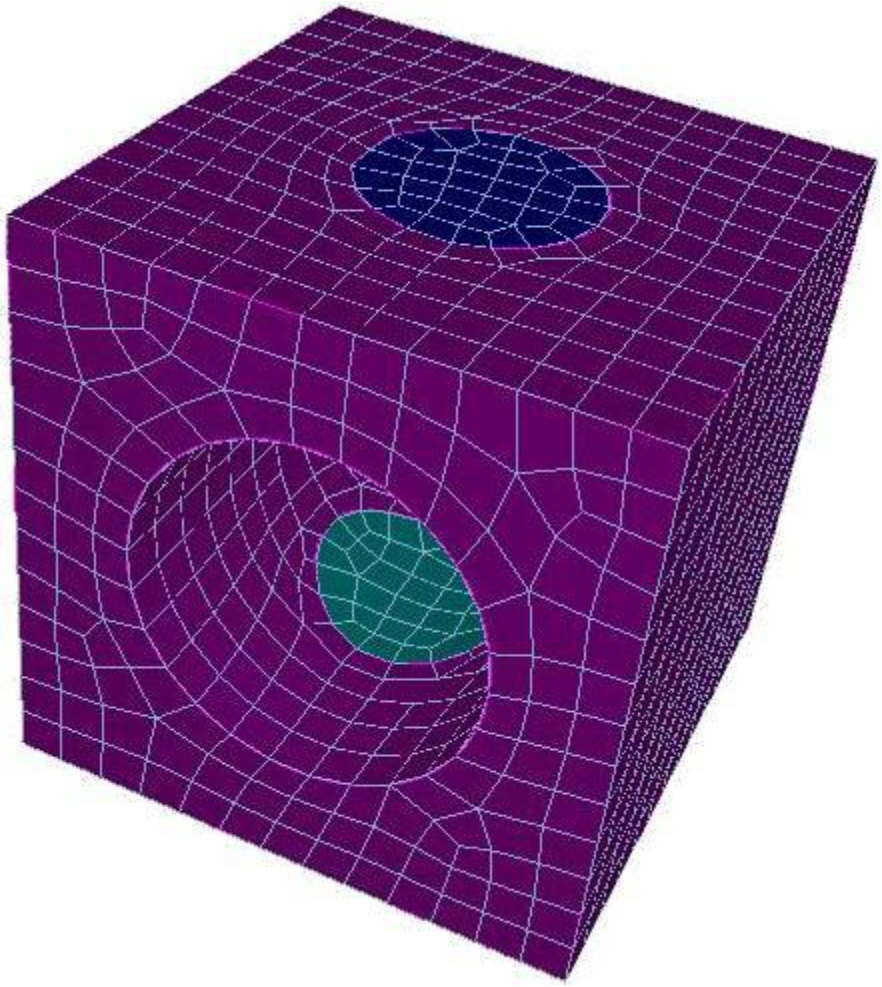


Figure 4: The mesh after meshcutting

Step 4: Final step. Figure 5 shows the final mesh after the mesh of the mesh of the two extra volumes is deleted. The commands are:

cubit> delete mesh volume 5 6 propagate
cubit> draw volume 4

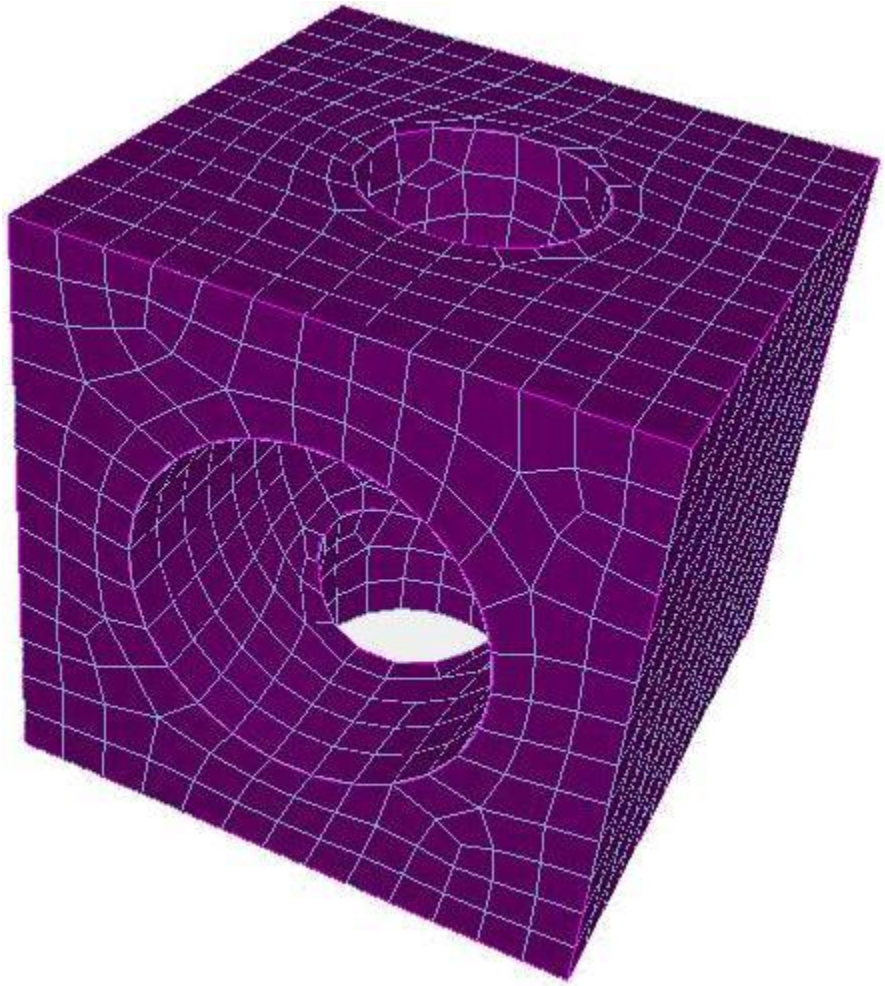


Figure 5: Final mesh after deleting unneeded elements

Collapsing Mesh Edges

CUBIT currently offers several options for modifying an existing finite element mesh. In addition to providing for coarsening and refining of hexahedral and triangle meshes, CUBIT can also reposition nodes by smoothing or by moving individual nodes.

The collapse edge command is also provided for making small modifications to an existing triangle mesh.

meshedit collapse edge <range>

This command will collapse the two triangles associated with the given edge, effectively removing the triangles from the mesh. This command only works on surface meshes, and only with triangles. If volumetric elements, or quads, are attached to the edge, the command does nothing to the mesh.

Mesh Grafting

Grafting is used to merge a meshed surface with a dissimilar unmeshed surface. In the process, the location of the nodes on the meshed surface will be adjusted to fit to the bounding curves of the unmeshed surface and the connectivity of the original mesh may be changed to improve the final quality of the mesh. This allows an unmeshed volume to be attached--or grafted--onto a meshed volume. Grafting is particularly useful for models that have intersecting sweep directions (see example below).

The command syntax for grafting is:

Graft Mesh Surface <id> [Volume <id>] onto Surface <id_range> [no_refine] [no_smooth]
Graft Mesh Volume <id> onto Volume <id> [no_refine] [no_smooth]

The first command will check that the first surface (meshed surface) is indeed meshed and that it overlaps the second surface. The mesh will then be adjusted and after any needed imprinting is done the two surfaces will be merged together.The second command searches for overlapping surfaces of the two given volumes. The first volume must be meshed. Any overlapping surface pairs that are found are grafted together in a similar sequence as given above. The difference between the two commands is that the second automatically determines the surfaces that should be grafted together.

NOTE: because of tolerances used to find overlapping surfaces, the second command may sometimes fail to find all overlapping surfaces between the two volumes. In this case the first command should be used to explicitly specify the surfaces to be grafted together.

Grafting Options

[no_refine]: This option tells grafting not to modify the connectivity of the original mesh. The mesh is still adjusted to fit the boundary of the branch surface but no new elements are added.

[no_smooth]: This option tells grafting not to perform the final smoothing of the modified surface or volume mesh.

Grafting Scope

The following is a list describing the current scope and limitations of grafting:

- Grafting only works on volumes meshed with hex elements.
- The unmeshed branch surface cannot have any point outside the boundary of the meshed trunk surface.
- Grafting may have difficulty with branch surfaces that are very thin with respect to the element size of the meshed surface or that have sharp angles.
- If grafting fails some of the nodes of the original mesh may have been moved. Check the mesh quality and re-smooth if needed.

Grafting Example

This example shows the four basic steps of grafting:

1. Partition the geometry (optional).
2. Mesh the trunk volume.
3. Graft the branch volume onto the trunk volume.
4. Mesh the branch volume.

Step 1: Partition the geometry

Figure 1 shows the model that will be meshed. The arrows in the figure show the two intersecting sweep directions. Figure 2 shows the model decomposed for grafting.

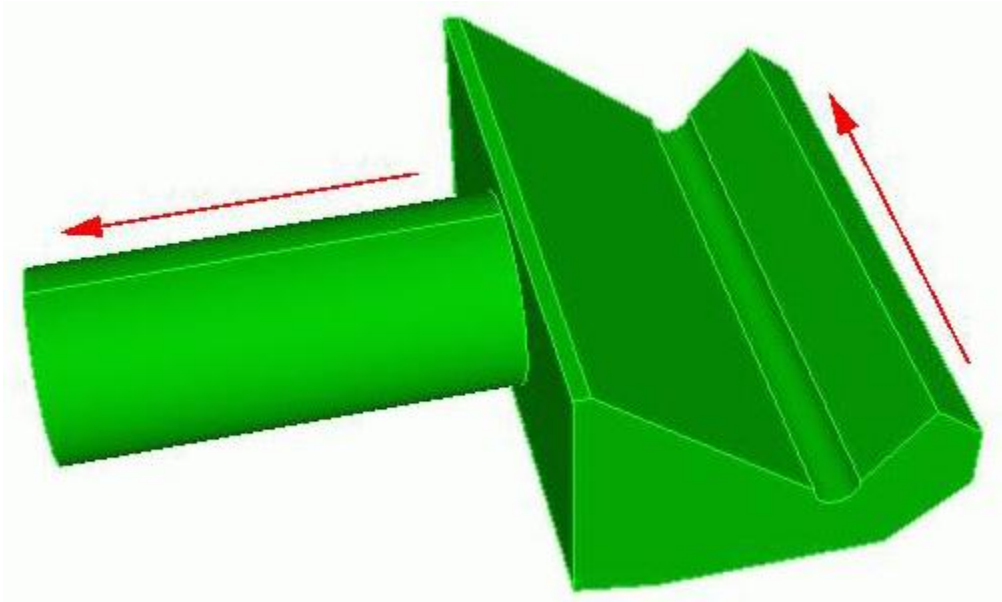


Figure 1. A model with two intersecting sweep directions.

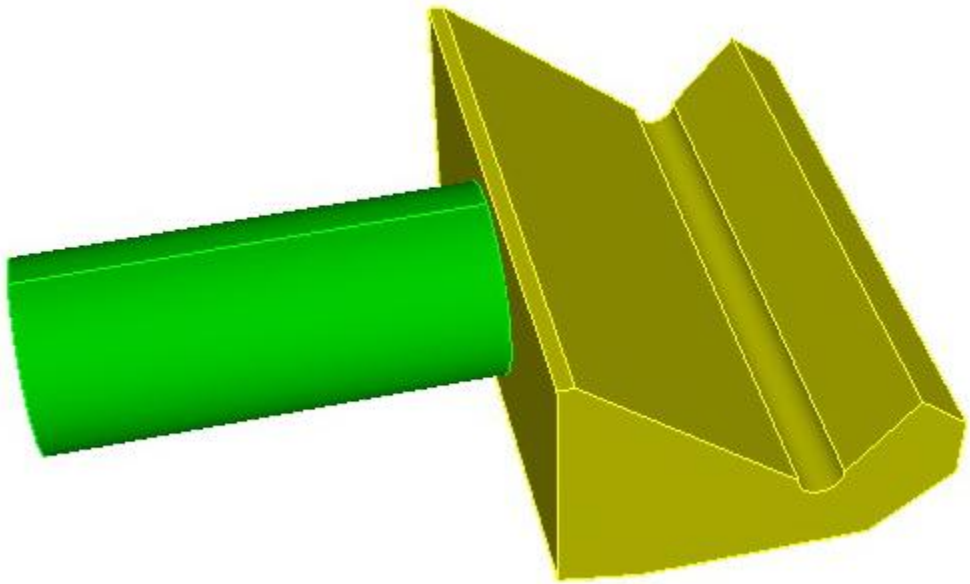


Figure 2. The model decomposed for grafting

Step 2: Mesh the trunk volume.

Figure 3 shows the mesh of the trunk volume. At this point the mesh on the trunk surface adjacent to the branch surface is a structured mesh that does not align with the boundary of the branch surface. The trunk and branch surfaces are two separate surfaces.

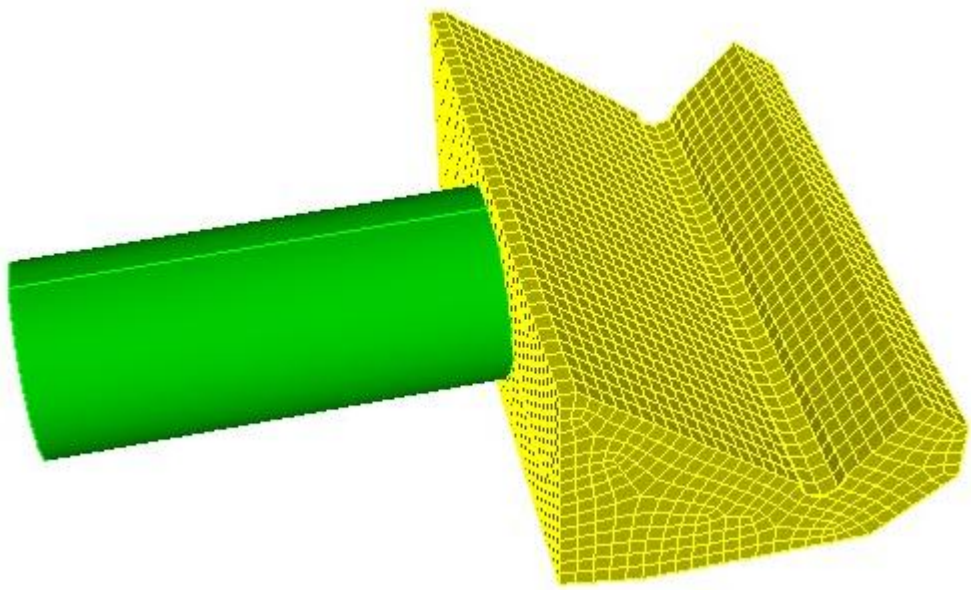


Figure 3. Meshed trunk volume.

Step 3: Graft the branch onto the trunk

Figure 4 shows the trunk surface after it has been modified to fit the branch surface. At this point the two surfaces have been merged together.

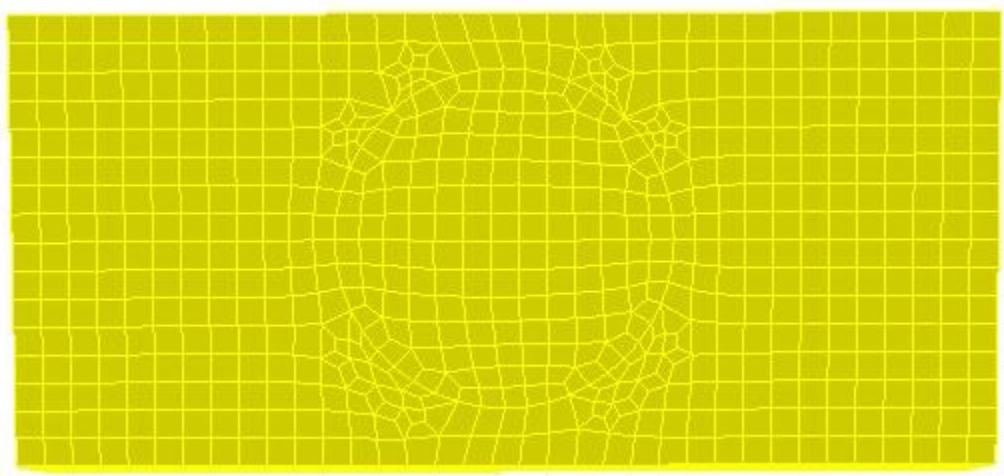


Figure 4. Trunk surface after grafting.

Step 4: Mesh the branch volume.

The final mesh is shown in Figure 5.

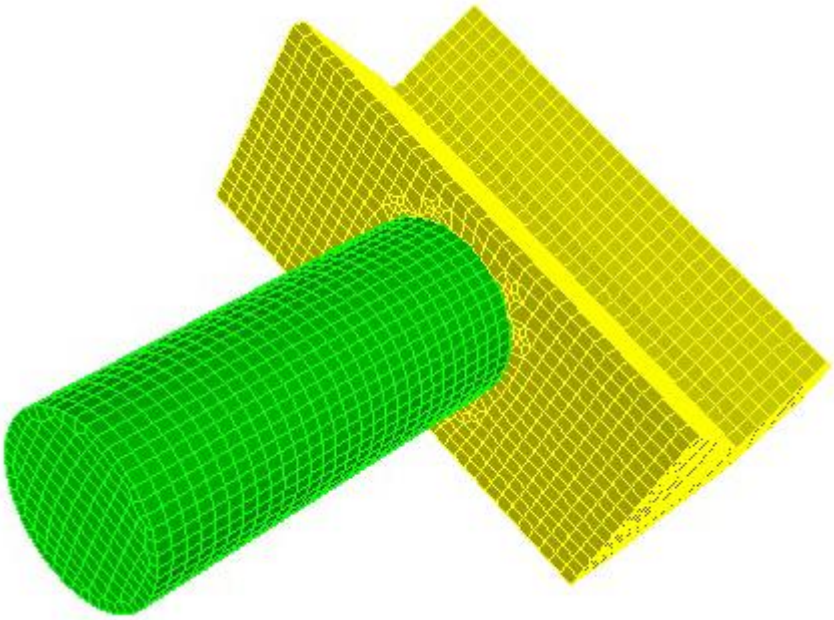


Figure 5. Final mesh

Node and Nodeset Repositioning

A capability to reposition nodesets and individual nodes is provided. This capability will retain all the current connectivity of the nodes involved, but it cannot guarantee that the new locations of the moved nodes do not form intersections with previously existing mesh or geometry. This capability is provided to allow the user maximum control over the mesh model being constructed, and by giving this control the user can possible create mesh that is self-intersecting. The user should be careful that the nodes being relocated will not form such intersections.

The user can reposition nodes appearing in the same nodeset using the **NodeSet Move** command. Moves can be specified using either a relative displacement or an absolute position. The command to reposition nodes in a nodeset is:

Nodeset <nodeset_list> move <delta_x> <delta_y> <delta_z>

Nodeset <nodeset_list> move to <x_pos> <y_pos> <z_pos>

The first form of the command specifies a relative movement of the nodes by the specified distances and the second form of the command specifies absolute movement to the specified position.

Individual nodes can be repositioned using the **Node Move** command. Moves are specified as relative displacements. The command syntax is:

Node <range> Move <delta_x> <delta_y> <delta_z>

Node <range> Move {[X <val>] [Y <val>] [Z <val>]}

Nodes can also be repositioned using a location specification. See Location, Direction, and Axis Specification for details on the location specification. The command syntax is:

Node <range> Move Location <options>

See also Transforming Mesh Coordinates.

Mesh Validity

After a mesh is generated, it is checked to ensure that the mesh has valid connectivity. If an invalid mesh is formed, then CUBIT automatically deletes it. This default behavior can be changed with the following command:

Set Keep Invalid Mesh [on|off]

The current behavior can be viewed with the following command:

List Keep Invalid Mesh

The Jacobian quality metric is also computed automatically to check quality after a mesh is generated. If the quality is poor, a warning is printed to the terminal.

Mesh Adaptivity and Sizing Functions

CUBIT provides several options for controlling the density of a mesh by adapting to various geometric, analysis, or user-defined properties. Interval sizes are defined automatically, explicitly, or through *sizing functions*. The sizing functions can be based on the physical features of the model, a previous analysis solution, or a user-specified bias. Adaptivity can apply to meshing either curves or surfaces.

Adaptive Curve Meshing

CUBIT provides several ways to adaptively mesh curves. Three curve meshing schemes are provided for this purpose. They include the following schemes:

- Curvature
- FeatureSize
- Stride

The first two schemes use characteristics of the geometric model to define element sizes. The third scheme uses a field function typically defined from a previous analysis solution. FeatureSize and Stride are alpha features and should be used with caution.

Adaptive Surface Meshing

Adaptive surface meshing in CUBIT produces a function following mesh which sizes elements based on the value of the driving function at the spatial location at which the element is to be placed. Adaptive surface meshing is performed using the paving, triadvance or tridelaunay algorithms in combination with an appropriate sizing function. The types of sizing functions that can be used are

- Bias Sizing
- Constant Sizing
- Curvature Sizing
- Linear Sizing
- Interval Sizing
- Inverse Sizing
- Super Sizing
- Test Sizing
- Exodus-based field function
- Geometry Adaptive (Skeleton Sizing)

Super sizing and test sizing functions are alpha features and should be used with caution.

The procedure for adaptively meshing a surface is to designate paving, triadvance or tridelaunay as the mesh scheme for that surface, assign sizing function types, and mesh the surface.

The command syntax of these commands is:

Surface < id > Scheme {Pave|TriAdvance|TriDelaunay}

then

Import Sizing Function '<exodusII_filename>' Block <block_id> Variable '<variable_name>' Time <time> [Deformed]

Surface <id> Sizing Function [Type] Exodus [Min <min_value> Max <max_value>]

or

Surface <id> Sizing Function [Type] {Constant|Curvature|Interval|Inverse|Linear|Super|Test|None}]

or

Surface <id> Sizing Function [Type] Bias Start Curve <id_range> {Finish Curve <id_range>| Factor <val>}

then

Mesh Surface <id>

Adaptive Volume Meshing

Adaptive volume meshing in CUBIT produces a function following mesh that sizes elements based on the value of the driving function at the spatial location at which the element is to be placed. Adaptive volume meshing is performed using the tetmesh scheme in combination with an appropriate sizing function. The types of sizing functions that can be used are constant, test and geometry adaptive. Test sizing is an alpha feature and should be used with caution. Other sizing functions will be added in future versions of Cubit.

The procedure for adaptively meshing a volume is to designate **tetmesh** as the mesh scheme for that volume, assign sizing function types, and mesh the volume.

The command syntax of these commands is:

Volume <id> scheme tetmesh
Volume <id> Sizing Function [Type] {Constant|Test|None}
Mesh Surface <id>

The following sections describe details of the various volume sizing methods.

- Constant Sizing
- Test Sizing
- Geometry Adaptive Sizing

Geometry Adaptive Sizing Function (Skeleton Sizing)

The **Geometry Adaptive Sizing Function** , also referred to as **Skeleton Sizing Function**, automatically generates a sizing function based on the geometric complexity of surfaces, solids, and assemblies. Regions of higher complexity will have a finer mesh size, while regions of relatively low complexity will have a coarser mesh size. Unlike other sizing functions in Cubit, the skeleton sizing function measures the geometric complexity while generating the mesh size. The framework consists of a background octree grid for storing the sizing function, a set of source entities that provide sizing information based on geometric complexity, and an interpolation module for calculating the size at the grid nodes of the background octree using the source entities. The background octree is generated using graphics facets and source entities. Source entities are generated using various tools that capture the geometric complexity of input CAD models. The tools used are: a disconnected 3D-skeleton to measure 3D-proximity, a disconnected 2D-skeleton to measure 2D-proximity, minimum principal radius of curvature to measure surface curvature, radius of curvature to measure curve curvature, and curve length to measure 1D-proximity. The source entities control the size and the sizing gradient at grid nodes that fall inside the scope of the source entities. An interpolation scheme is used to ensure that maximum and minimum size falls within the user specified limits and that the maximum gradient is also within the user specified limit. During mesh generation, the size at each point is linearly interpolated using the size at the nodes of the octree cell containing that point. When the skeleton sizing function is called on an entity, it is also applied to all of its sub-entities.

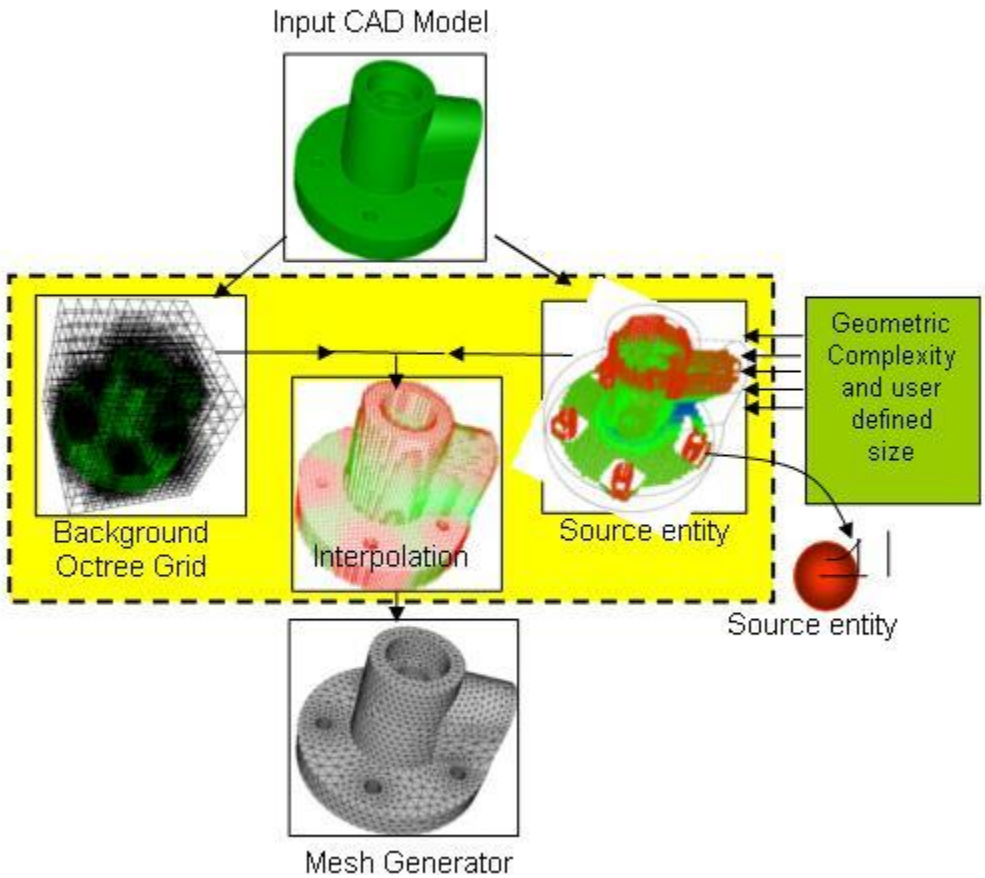


Figure 1: Overview of Computational Framework

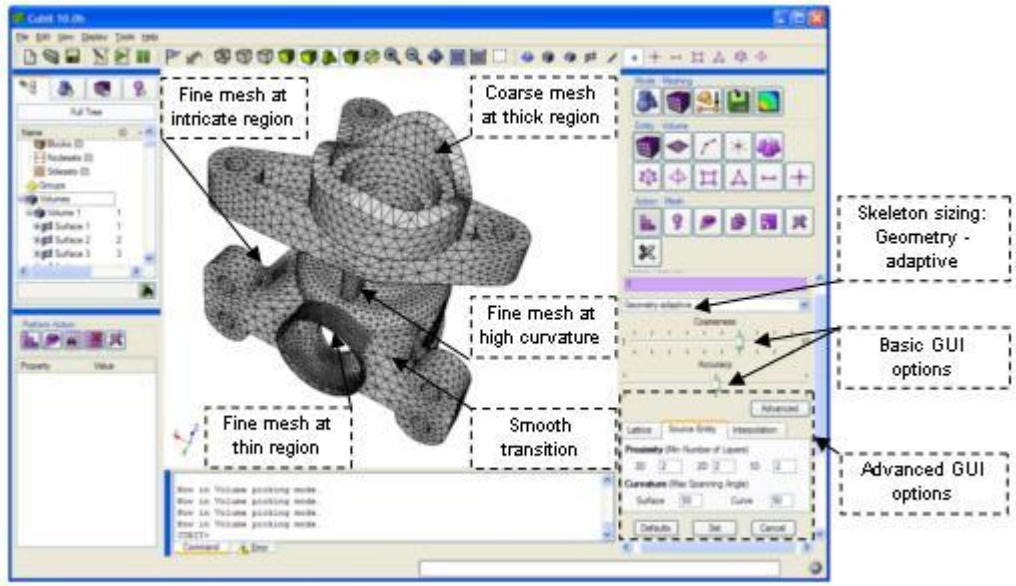


Figure 2:GUI for skeleton sizing function

Command Line Syntax

Skeleton sizing on multiple surfaces:

Surface <surf_id_range> Sizing Function [Type] Skeleton [scale <1 to 10 = 7>] [time_accuracy_level <1 to 3 = 2>]

[min_depth <3 to 8 = 5>] [max_depth <4 to 9 = 7>]

[min_num_layers_2d < 1 to N = 1>] [min_num_layers_1d < 1 to N = 1>] [max_span_ang_surf <5.0 to 75.0 = 45.0 degrees>] [max_span_ang_curve <5.0 to 75.0 = 45.0 degrees>] [min_size <float>] [max_size <float>] [max_gradient <float=1.5>] [skl_interpolate_scheme <0 to 5 = 2>]

Skeleton sizing on single volume:

Volume <vol_id> Sizing Function [Type] Skeleton [scale <1 to 10 = 7>] [time_accuracy_level <1 to 3 = 2>]

[min_depth <3 to 8 = 5>] [max_depth <4 to 9 = 7>]

[min_num_layers_3d < 1 to N = 1>] [min_num_layers_2d < 1 to N = 1>] [min_num_layers_1d < 1 to N = 1>] [max_span_ang_surf <5.0 to 75.0 = 45.0 degrees>] [max_span_ang_curve <5.0 to 75.0 = 45.0 degrees>] [min_size <float>] [max_size <float>] [max_gradient <float=1.5>] [skl_interpolate_scheme <0 to 5 = 2>]

The options are explained below:

Basic Arguments

- **scale** (default 7): The overall size of the elements is controlled by this argument. A coarser mesh can be generated by increasing the value of scale up to 10.0. To get a finer mesh, decrease the value of the scale (minimum value = 1).
- **time_accuracy_level** (1 to 3, default 1): This controls the computational time and accuracy level by adjusting various internal parameters of skeleton sizing function. Users should try levels in increasing order. Level 1 takes the shortest time to compute the skeleton sizing function and Level 3 takes the longest time to compute the skeleton sizing function. However, Level 1 is less accurate than Level 2 and Level 3.

Advanced Arguments

Lattice Arguments:

- **min_depth** (default = 5): min_depth controls the maximum cell dimension of the background octree grid. (PR-Octree, where PR:Point Region). The higher the value of min_depth, the smaller the dimension of the maximum-sized cell. Computational time increases with increasing min_depth.
- **max_depth** (default = 7): max_depth controls the minimum cell dimension. If the object contains very fine features then increasing the value of max_depth is suggested. The maximum depth has been limited to 9.

Note: These arguments override the basic arguments. For example, time accuracy level 1 internally sets min_depth = 4 and max_depth = 6, and when min_depth is set to 4 and max_depth is set to 7 in the advanced options (recommended for models with fine features), then advanced options override the basic options.

Source Entity Arguments

- **min_num_layers_3d** (Any value greater than 1, default 1): This parameter ensures that a minimum specified number of layers exist across the thickness of the volume. This parameter could be useful in generating meshes for mold flow simulation.
- **min_num_layers_2d** (Any value greater than 1, default 1): This parameter ensures that a minimum specified number of layers exist across the thickness of a surface.
- **min_num_layers_1d** (Any value greater than 1, default 1): This ensures that any curve contains a minimum of specified number of intervals.
- **max_span_ang_curve** (Range 5.0 to 75.0, default 45 deg): Maximum spanning angle is a parameter that controls the mesh size at curved regions of curves. It is defined as the angle subtended by the normals at the end nodes of the mesh edge in the curved region of a curve. When finer mesh is needed at curved region of curves, then max_span_ang_curve should be decreased.
- **max_span_ang_surf** (Range 5.0 to 75.0, default 45 deg): Maximum spanning angle is a parameter that controls the mesh size at curved regions of surfaces. It is the angle subtended by the normals at the end nodes of the mesh edge in a curved region of a surface. When finer mesh is needed at curved region of surfaces, then max_span_ang_surf should be decreased.

Interpolation Arguments

- **max_size** (default=auto): The value for max_size is calculated automatically by default. Users can specify any positive real number based on the dimensions of the model to control the max size of the elements. If the skeleton sizing function creates large elements, than this argument can be used to control the maximum size of the element.
- **min_size** (default=auto): The value for this min_size is calculated automatically by default. Users can specify any positive real number based on dimension of the model to specify the minimum size of the elements.
- **max_gradient** (1 to 3, default=1.25): The transition in element size is controlled using this parameter. The larger the value, the less gradual the transition, which reduces the number of elements.

Skeleton with Other Sizing Controls

Skeleton sizing function produces a smooth sizing function when called with other sizing controls available in Cubit. Skeleton sizing function behaves as SOFT firmness level. Skeleton sizing function always respects interval count specified on the curves. Skeleton sizing function respects interval size on curves and surfaces only if it is specified after calling the skeleton sizing function.

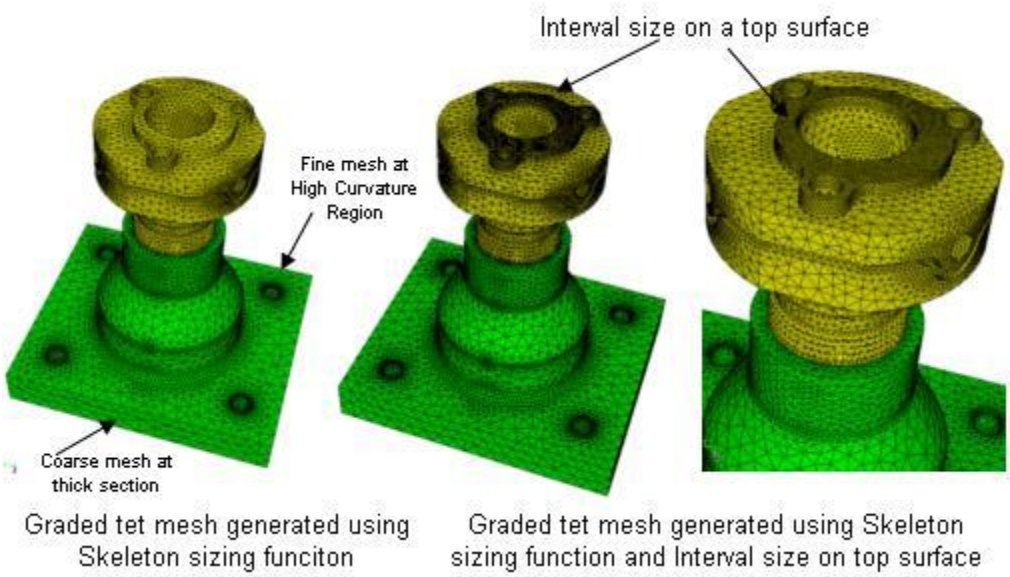


Figure 3: Skeleton sizing function with other sizing controls

Limitations

- Currently, the skeleton sizing function is only intended for use with ACIS models. Sizing function generation with other geometry engines in Cubit such as Granite/ProE and facet-based (STL, facet, and MBG format) is not guaranteed or supported in Cubit 10.0.
- The skeleton sizing function is mainly tested with trimesh and tetmesh schemes. In general, structured or semi-structured meshing schemes do not have enough flexibility to utilize the skeleton sizing function. It is recommended that the skeleton sizing be used only with unstructured meshing schemes.
- Sizing function generation of assemblies is also not guaranteed in Cubit 10.0. When used with assemblies, time_accuracy_level 2 is strongly recommended. For example, “vol all sizing function skeleton time_accuracy 2”

Bias Sizing Function

Syntax:

Surface <id> Sizing Function Type Bias Start Curve <id_range>
{Finish Curve <id_range>| Factor <val>}

Synopsis:

The **Bias** sizing function for surfaces is similar to biasing curves. Indeed, setting a bias sizing function for a surface will bias the boundary curves, as well as control paving to follow the bias inside the surface. You first specify the size of a couple of bounding curves (the start curves), then specify the bias sizing function for the surface.

Discussion:

Recall that for biasing curves, you specify the start and end vertex. For the bias sizing function, you specify the start curves, from which to bias away. The sizes of these curves should already be set before setting the surface sizing function since their average size is taken to be the starting size (almost). If the start curve sizes change, then you should set the surface sizing function again.

You can either supply a geometric factor, or the set of finish curves whose sizes you want to match at that distance. A geometric factor. It automatically sizes and biases or dualbiases the non-start curves, including any finish curves. These curves need not be perpendicular to the starting curves. The interval count and scheme are soft-set, so they won't be changed if they are already hard-set. If the size of the start curves or finish curves are changed, then the sizing function command should be re-issued.

The sizing function value at a point is defined in terms of the straight-line distance from the point to the closest starting curve. So, it works best if all the starting curves have the same size, and the surface is relatively flat. But, starting curves need not be parallel to one another. Similarly, the non-start curves need not have any particular orientation wrt the start curves.

The bias sizing function was designed to easily set the sizes of a sequence of adjoining surfaces: assign a size to the curve you want to bias away from, then set the bias sizing function of the first surface, with its finish curves being the start curve of the second surface, etc. See the last example below.

Examples:

Here are some example journal files and resulting pictures:

```
# bias_sz_fn_demo.jou
brick x 100 y 10 z 10
color vol 1 red
surface 1 scheme pave
surface all except 1 visibility off
# label curve interval
# graph text 2
display

# mesh 1
curve 4 size 2
surface 1 sizing function type bias start curve 4 factor 1.3
mesh surface 1
# see figure 1
```

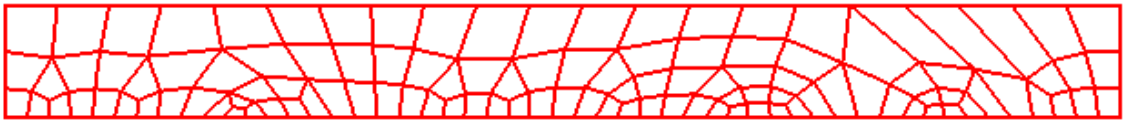


Figure 1. Surface with bias sizing function factor > 1.

```
# mesh 2
delete mesh
surface 1 sizing function type bias start curve 4 factor {1/1.1}
mesh surface 1
# see figure 2
```

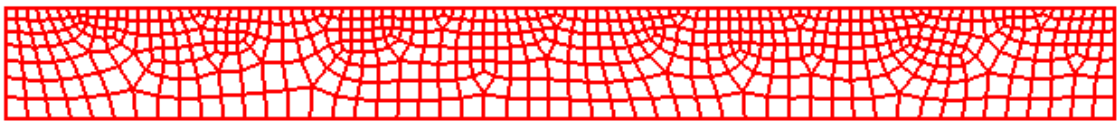


Figure 2. Surface with bias sizing function factor < 1

```
# mesh 3
reset
cyl rad 6 z 1
cyl rad 4 z 1
sub 2 from 1
section body 1 yplane
section body 1 xplane
surf all except 19 vis off
color vol 1 red
display
```

```
# finish curve mesh
surf 19 scheme qtri base scheme pave
surface 19 size 0.7
curve 26 size 0.07
surface 19 sizing function type bias start curve 26 finish curve 25
mesh surface 19
pause
# see figure 3
```

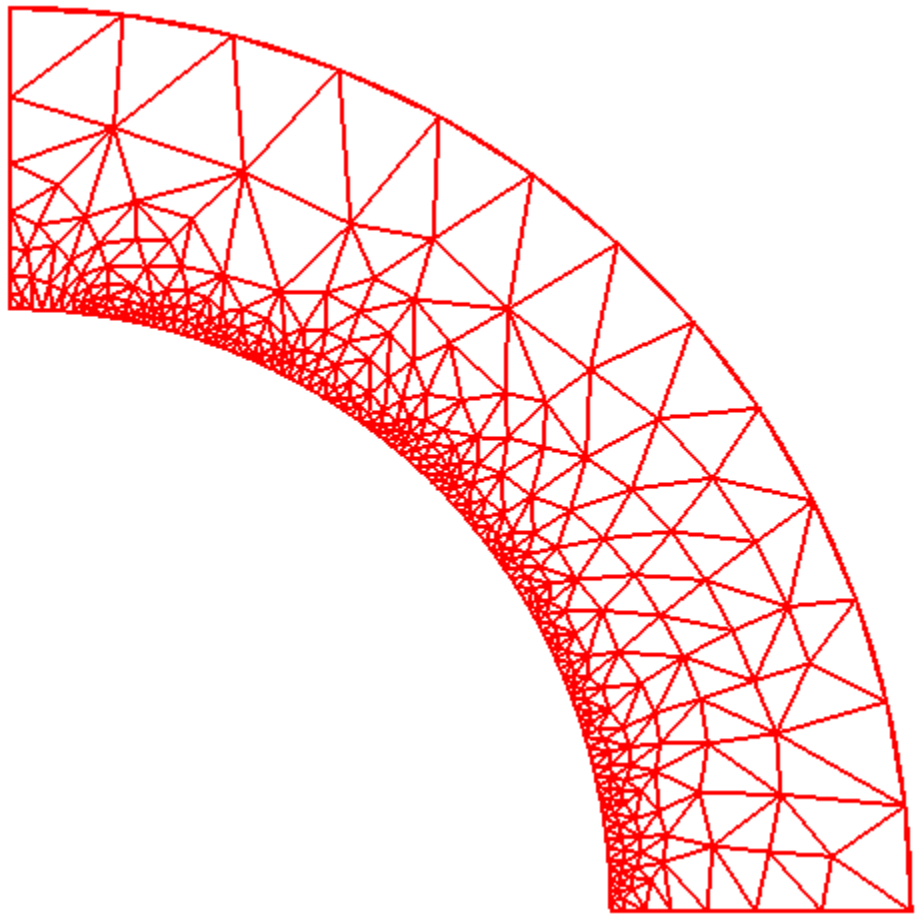


Figure 3. Surface with bias sizing function start and finish curve. Scheme qtri, base scheme pave.

```
# dual bias mesh
delete mesh
curve 25 26 size 0.02
curve 25 26 scheme equal
surface 19 sizing function type bias start curve 26 25 factor 1.3
mesh surface 19
zoom curve 12
pause
# see figure 4
```

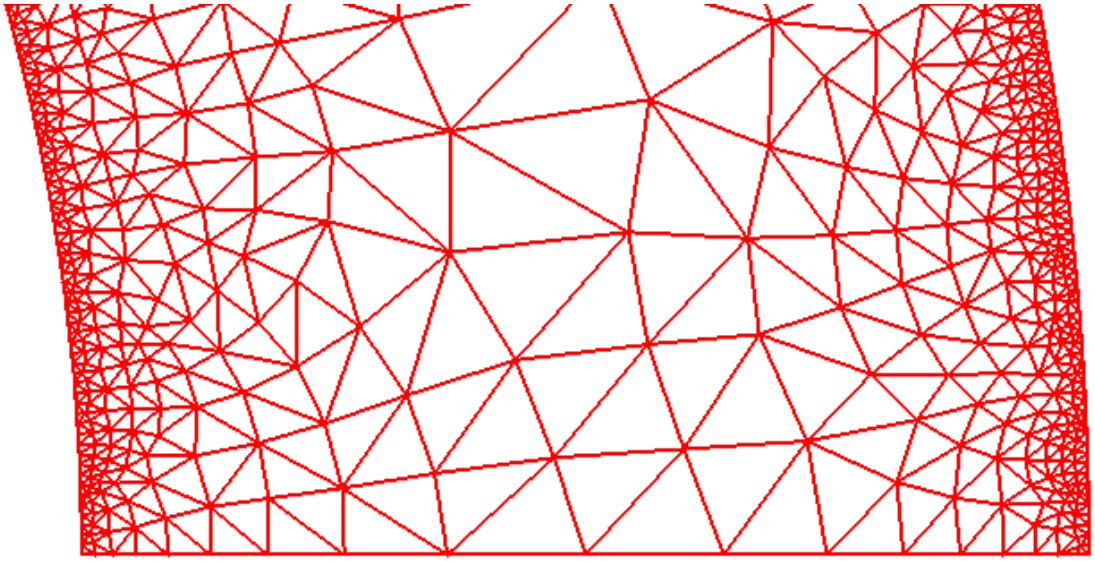


Figure 4. Close up of surface with dual bias sizing function start and finish curve. Scheme qtri, base scheme pave.

```
# funny face
reset
prism sides 5 z 1 radius 1
cylinder radius 0.1 z 1
body 2 move -0.4 0 0
subtract 2 from 1
cylinder radius 0.1 z 1
body 3 move 0.2 0 0
subtract 3 from 1
prism sides 6 radius 0.2 z 1
body 4 move 0 -0.4 0
subtract 4 from 1
surface all except 34 visibility off
color vol 1 red
display
surface 34 scheme pave
curve 23 19 size 0.01
surface 34 sizing function type bias start curve 19 23 factor 1.3
mesh surface 34
# see figure 5
```

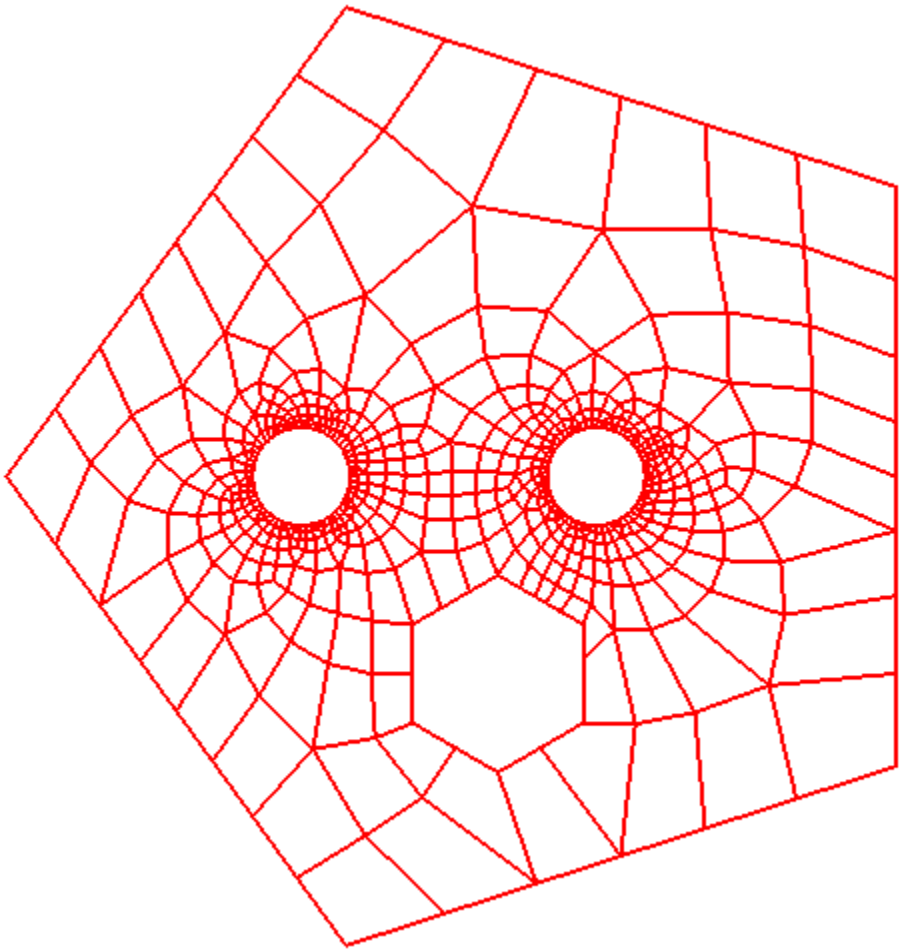


Figure 5. Bias away from two round holes.

```
# bias surface chain
reset
cylinder radius 1 z 1
cylinder radius 0.2 z 1
cylinder radius 0.4 z 1
cylinder radius 0.8 z 1
imprint body all
delete body 2 3 4
section body 1 xplane
section body 1 yplane
surface all except 42 43 44 45 vis off
color volume 1 red
surface all scheme pave
curve 55 interval 36
surface 43 sizing function type bias start curve 55 factor 1.3
surface 44 sizing function type bias start curve 57 factor 1.3
# curve 57 had its size determined by a prior bias sizing function
surface 45 sizing function type bias start curve 58 factor 1.3
surface 42 sizing function type bias start curve 55 factor 1.3
mesh surface 42 43 44 45
display
highlight curve in surface 42 43 44 45
# see figure 6
```

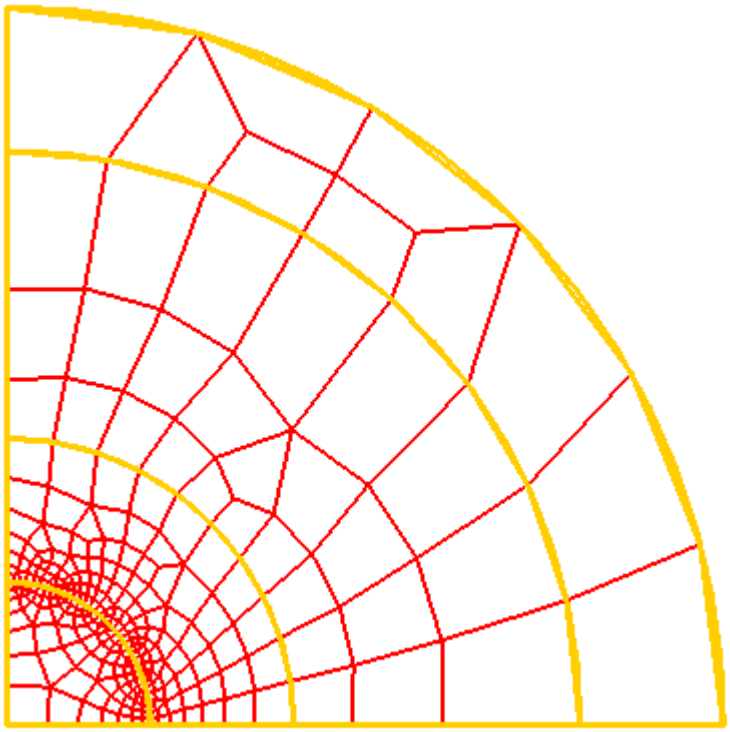


Figure 6. A chain of biased surfaces. Only one curve's intervals were explicitly set.

Constant Sizing Function

Syntax:

Surface <id> Sizing Function [Type] Constant [<val>]

Volume <id> Sizing Function [Type] Constant [<val>]

Synopsis:

The **Constant** sizing function specifies an element size that is constant over the interior of the surface or volume. If **<val>** is omitted, the element size is set to the average interval size of the bounding curves for surfaces or the average element size of the bounding surfaces for volumes.

Curvature Sizing Function

The **Curvature** sizing function determines element size based on the curvature evaluation of a surface at the current location. Two surface curvature values (taken perpendicular to each other) are compared at the location of interest, and the largest is used as the sizing function for the mesh. Figure 1 shows a solid with a highly deformed surface which displays rapid change of surface curvature at several locations.

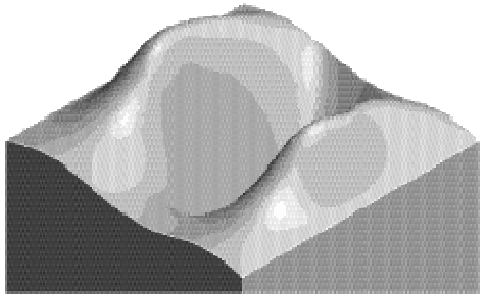


Figure 1. NURB solid with high surface curvature change

Figure 2 depicts a normal paved mesh of this surface using a common size on all bounding curves and no sizing function in the interior. The total number of quadrilateral shell elements for this case is 1988. Figure 3 shows a mesh which was generated with the curvature sizing function option. The mesh is graded denser in the regions of quickly changing curvature, such as at the tops of the hills and at the bottom of the valley. Due to the intense interrogation of the underlying geometric modeler which the curvature method relies on, this option can be very computationally expensive.

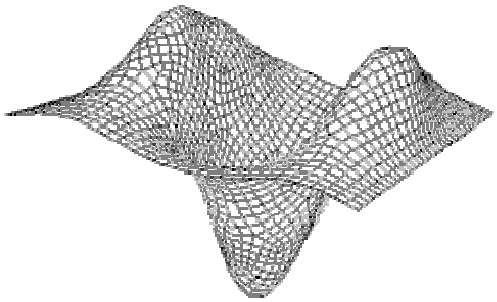


Figure 2. NURB mesh with no interior sizing function

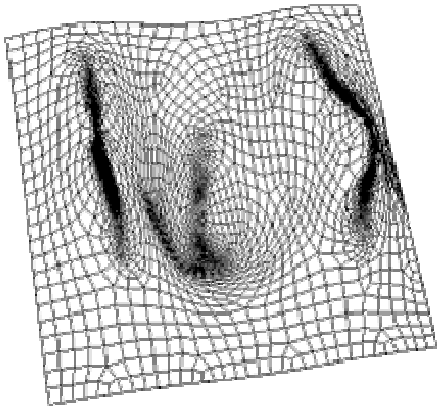


Figure 3. NURB mesh with curvature sizing function

Linear Sizing Function

The **Linear** class of sizing functions determines element size based on a weighted average of edge lengths for mesh edges bounding the surface being meshed. There are several variants of this class of sizing function. The Linear function bases edge length at a location on the lengths of edges bounding the surface weighted by their inverse distance from the current location. The result of this weighting is a more gradual change in mesh density during a transition between dense and coarse mesh. Figure 1 shows the same NURB surface mesh but with intervals of 34 on two curves and intervals of 16 on the remaining two bounding curves and no sizing function. It can be observed that the mesh progresses more rapidly inward from the coarser meshed curves, which locates the transition region much closer to the finer meshed curves. To combat this, the Linear function weights the sizing of new elements such that these transitions occur slower. Figure 2 displays two views of the same NURB geometry with the same bounding curve mesh density using the linear sizing function.

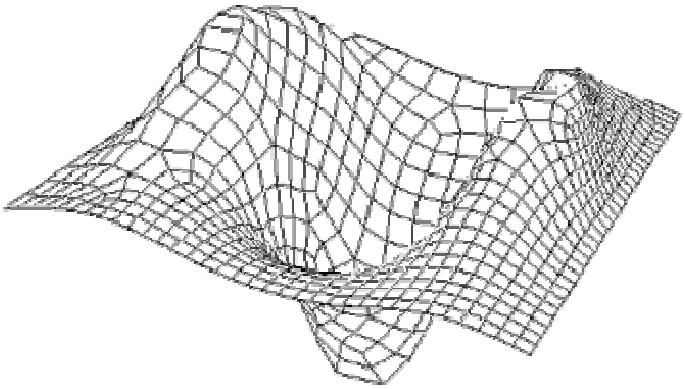
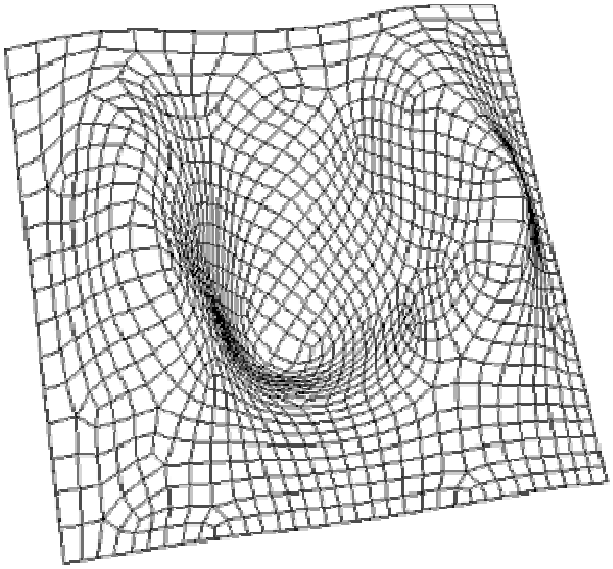


Figure 1. NURB mesh with no sizing function, 34 by 16 density



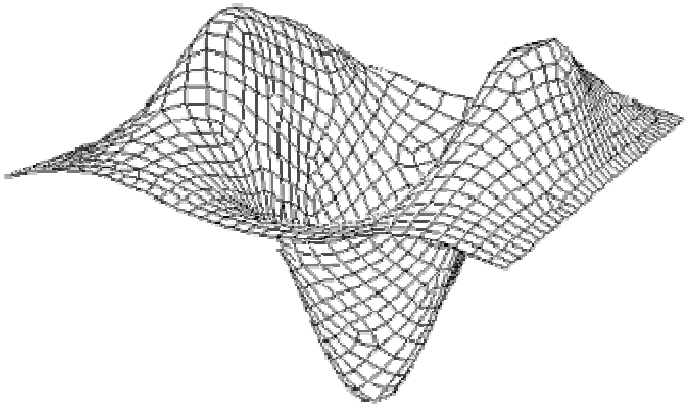
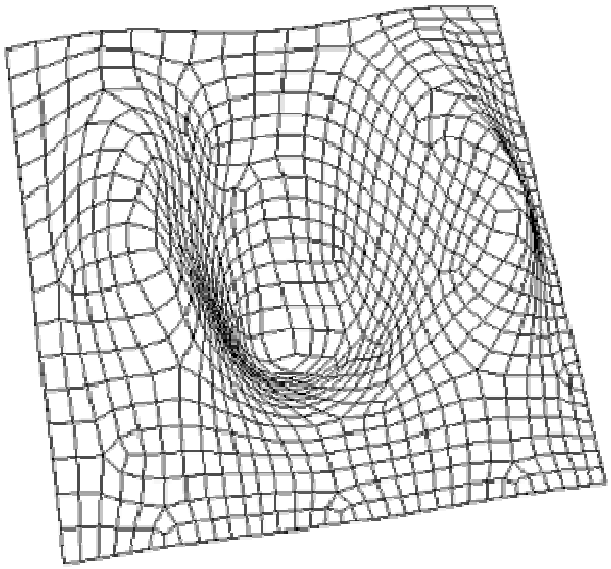


Figure 2. NURB mesh with linear sizing function, 34 by 16 density

Interval Sizing Function

The **Interval** sizing function is similar to the Linear function, but bases edge length at a location on the squared lengths of edges bounding the surface weighted by their inverse distance from the current location. An example is shown below.



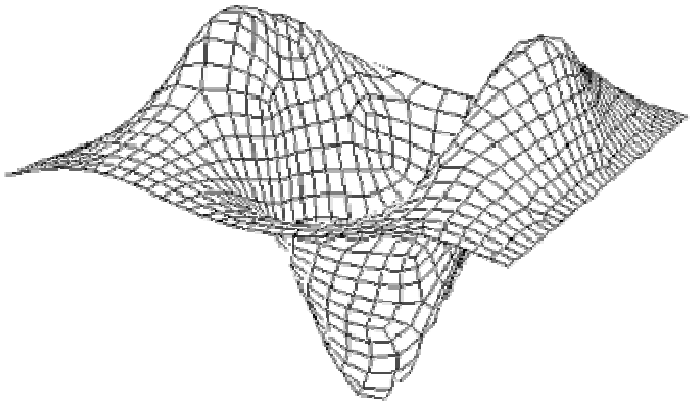
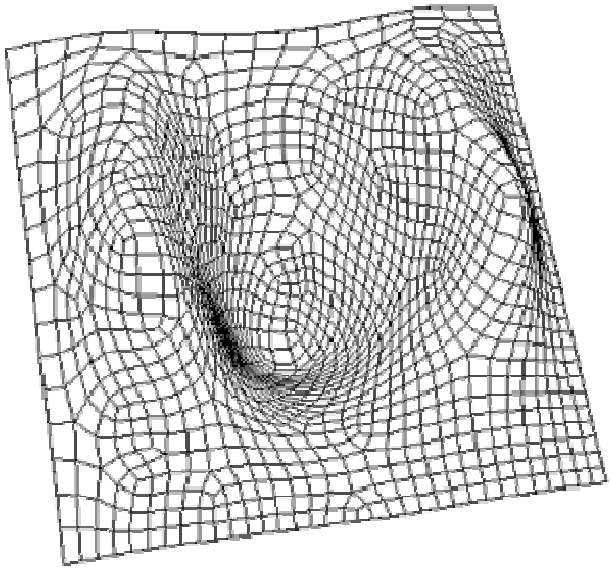


Figure 1. NURB mesh with interval sizing function, 34 by 16 density

Inverse Sizing Function

The Inverse sizing function is also similar to the Linear function, but this method bases edge length at a location on the inverse lengths of edges bounding the surface weighted by their inverse distance from the current location (see Figure 1). The difference between the three linear sizing functions (Linear, Interval, Inverse) is sometimes subtle, but is driven by the geometry being meshed since the influence of these functions is strongly controlled by the number, positioning, and mesh density of the bounding curves relative to the interior surface area.



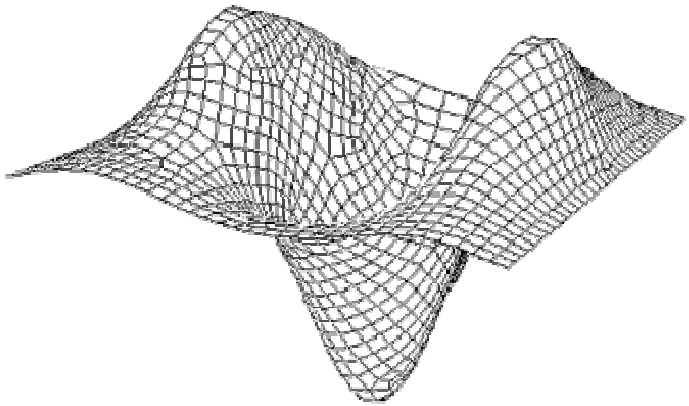


Figure 1. NURB mesh with inverse sizing function, 34 by 16 density

Exodus II-based Field Function

The ability to specify the size of elements based on a general field function is also available in CUBIT. With this capability the desired element size can be determined using a field variable read from a time-dependent variable in an Exodus file. Either node-based or element-based variables can be used. Importing a field function and associating it with a surface, and normalizing that function are done in two separate steps to allow renormalization without having to read the mesh in again. Currently, field functions are imported from element and node-based Exodus II data. Thus, a field function is a time-dependent element variable in an Exodus II file. The mesh block containing the corresponding elements must be imported along with the field function data. For details on the adaptive paving algorithm, see [Blacker, 90]. Exodus variable-based adaptive paving is accomplished in CUBIT in several steps:

1. Surface mesh scheme set to Pave. Bounding curve mesh schemes can also optionally be set to Stride.
2. An Exodus mesh and time-dependent variable for that mesh is read into CUBIT.
3. The mesh and variable data are associated to geometry.
4. The Exodus variable is normalized to give localized size measures, and the surface sizing function type is designated.
5. Surface is meshed.

The following command is used to read in a field function and its associated mesh:

Import Sizing Function '<exodusII_filename>' Block <block_id> Variable `<variable_name>' Time <time_val> [Deformed]

where **<block_id>** is the element block to be read, **<variable_name>** is the Exodus time-dependent variable name (either element-based or nodal-based), and **<time_val>** is the problem time at which the data is to be read, the **Deformed** keyword indicates whether deformation has been accounted for on the new model (for information on creating deformed 2D geometry from EXODUSII data, see Importing EXODUSII Files) and needs to be accounted for in the sizing function data. When this command is given, the nodes and elements for that element block are read in and associated to geometry already initialized in CUBIT.

Note that when a sizing function is read in, the mesh is stored in an ExodusMesh object for the corresponding geometry, and therefore the geometry is not considered to be meshed. Also note that if deformation is not being modelled, the geometry to which the mesh is being associated must be in the same state as it was when that mesh was written (see Mesh Importing and Duplicating for more details on importing meshes).

Once the field function has been read in and assigned to a surface, it can be normalized before being used to generate a mesh. The normalization parameters are specified in the same command that is used to specify the sizing function type for the surface. The syntax of this command is:

Surface < id > Sizing Function Type Exodus [Min <min_val> Max <max_val>]

If normalization parameters are specified, the field function will be normalized so that its range falls between the minimum and maximum values input. Subsequent normalizations operate on the normalized data and not on the original data. If an element-based variable is used for the sizing function, each node is assigned a sizing function that is the average of variables on all elements connected to that node. Nodal variables are used directly.

After the sizing function normalization, the surface can be meshed using the normal meshing command.

For example, the left image in Figure 1 depicts a plastic strain metric which was generated by PRONTO-3D [Taylor, 89] a transient solid dynamics solver, and recorded into an ExodusII data file. When the file is read back into CUBIT, the paving algorithm is driven by the function values at the original node locations, resulting in an adaptively generated mesh [Attaway, 93]. The right image in Figure 1 depicts the resulting mesh from this plastic strain objective function.

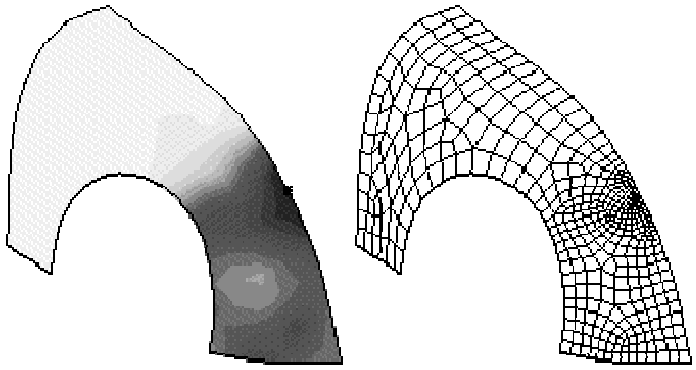


Figure 1. Plastic strain metric and the adaptively generated mesh

Curve Meshing with Exodus II - based Field Functions

In addition to the capability to adaptively mesh surface using a field function, curves may also be meshed separately using the Exodus II information. The Stride scheme for meshing curves is used for this purpose.

Mesh Deletion

Meshing a complex model often involves iteration between setting mesh parameters, meshing, and checking mesh quality. This often requires removing mesh, for only an entity or for an entity and all its lower order geometry, or sometimes for the entire model.

The command to remove all existing mesh entities from the model is:

Delete mesh

The command for deleting mesh on a specific entity is:

Delete mesh {geom_list} [Propagate]

These commands automatically cause deletion of mesh on higher dimensional entities owning the target geometry.

If the Propagate keyword is used, mesh on lower order entities is deleted as well, but only if that mesh is not used by another higher order entity. For example, if two surfaces (surfaces 1 and 2) sharing a single curve are meshed, and the command "delete mesh surface 1 propagate" is entered, the mesh on surface 1 is deleted, as well as the mesh on all the curves bounding surface 1 except the curve shared by surface 2. In some cases, the capability to delete individual mesh faces on a surface is needed. Deleting a mesh face involves closing a face by merging two mesh nodes indicated in the input. The syntax for this command is:

Delete Face <face_id> Node <node_id> [Node <diagonal_node_id>]

This command is provided primarily for developers' use, but also provides the user fine control over surface meshes. At the present time, this command works only with faces appearing on geometric surfaces and should be used before any hex meshing is performed on any volume containing the face to be deleted.

Importing and Exporting Files

- Importing Geometry
- Exporting Geometry
- Importing an Exodus II Mesh
- Exporting the Finite Element Model

This chapter describes methods for importing and exporting geometry and mesh data within CUBIT. CUBIT supports ACIS and Granite solid model geometry internally. Built-in translators can convert IGES and STEP files into ACIS or Granite geometry formats. CUBIT also supports mesh-based geometry. Facet or STL files can be imported as mesh-based geometry.

Mesh files can be imported as a free mesh, or associated with existing ACIS or mesh-based geometry in CUBIT. Finite element data is exported as Exodus II data files or ABAQUS files.

Importing Geometry

- Importing ACIS Models
- Importing FASTQ Models
- Importing STEP Files
- Importing IGES Files
- Importing Facet Files
- Importing Granite Models
- Other Formats

Internally, CUBIT represents geometry as either ACIS solid model geometry, Granite solid model geometry, or mesh-based geometry. CUBIT can import ACIS geometry in the native "sat" file format. CUBIT can also import STEP and IGES files and internally converts them into ACIS solid model geometry. For compatibility with Sandia legacy applications, CUBIT can import FASTQ input decks to create ACIS geometry, as well.

Other Geometry Formats

As shown above, Cubit currently natively supports ACIS geometry, Granite geometry, and mesh-based geometry. Other formats will need to be translated to one of these formats. Cubit has internal translators that will read STEP and IGES files. If you have geometry that has been created in another format, such as in SolidWorks, you will need to translate that geometry into something that Cubit can read. Many solid modeling packages have an Export ACIS .sat command, which is probably the easiest way of translating your model. If you do not have that option, there are some other possibilities.

- Try a different file format, such as STEP or IGES.
- Use 3dshare.com. This is a website hosted by Spatial, the creators of ACIS, that can translate most geometry formats into ACIS.
- As a last resort, contact the Cubit team. They might have other options for importing your file.

Importing ACIS Files

The command used to read an ACIS file is:

```
Import Acis '<acis_filename>' [no_bodies][no_surfaces] [no_curves][no_vertices][Group
{'<name>'|<id>}] [binary|ascii] [sort] [XML]
```

The **import ACIS** command is the primary mechanism for generating geometry within CUBIT. ACIS parts can be generated and saved with CUBIT, but in most cases are developed within a 3rd party CAD package and exported for use in CUBIT. CUBIT provides the capability to import ACIS solid models and make modifications to them so they can be meshed. CUBIT incorporates the commercial ACIS libraries developed and maintained by Spatial Inc. for reading and writing ACIS format files. IGES and STEP format files can also be imported and exported to/from CUBIT using the Spatial's libraries.

The following options can be used when importing an ACIS file.

[no_bodies][no_surfaces] [no_curves][no_vertices]

It is possible to include *free* entities (vertices, curves and surfaces) in the file. The default operation is to read all entities in the file whether they are included as part of a body or are free. By using any of the options **no_bodies, no_surfaces, no_curves,** or **no_vertices**, the user may exclude certain types of *free* entities.

[group]

The **group** option of the import command will allow the user to create a group for each set of imported geometry. The newly created group can later be accessed using the name or id specified with the group option.

[binary|ascii]

The import capability of ACIS files supports both the ASCII format (.sat) and binary format (.sab). When importing, the filename extension will determine the default file type, be it ASCII or binary. A (.sat) extension will default to ASCII, while a (.sab) extension will default to binary. If you use a different file extension you can specify the type with the **[binary|ascii]** option. Binary files can be significantly faster but are not guaranteed to be upward compatible, nor cross-platform compatible (although testing has determined compatibility between NT and HP/UX). Therefore, it is recommended that models be archived in ASCII format.

[sort]

Normally the numerical IDs of the geometric entities contained in the ACIS model are used directly within CUBIT. The sort option provides the capability to compress the IDs read from the ACIS file. The sort option does the same thing as the compress ids sort command, but combines it with the **import** command to remove a step in the process.

[XML '<xml_filename>']

This option will read assembly information and other metadata from an XML file in the DART metadata XML format. See the metadata documentation and the Analyst's Home Page for details.

Importing ACIS files at startup

ACIS files can also be imported using the **"-solid"** option when starting CUBIT from the UNIX command prompt. (See Execution Command Syntax for details.) Note that the filename must be enclosed in single or double quotes. This command will create as many bodies within CUBIT as there are bodies in the input file.

See also Exporting ACIS Files.

Importing FASTQ Files

CUBIT can read a FASTQ file and convert it into an ACIS model:

Import Fastq '<fastq_filename>'

Note that the filename must be enclosed in single or double quotes.

FASTQ is an older, 2d meshing tool; (Blacker 88.) FASTQ files are a series of commands much like a CUBIT journal file. All FASTQ commands are fully supported except for the "Body" command (it is unnecessary and ignored), the "corn" (corner) line type, and some of the specialized mapping primitive "Scheme" commands. Standard mapping, paving, and triangle primitive scheme commands are handled. The pentagon, semicircle, and transition primitives are not handled directly, but are meshed using the paving scheme. The FASTQ input file may have to be modified if the Scheme commands use any non-alphabetic characters such as `+', `(`, or `)'. Circular lines with non-constant radius are generated as a logarithmic decrement spiral in FASTQ; in CUBIT they will be generated as an elliptical curve.

Since a FASTQ file by definition will be defined in a plane, it must be projected or swept to generate three dimensional geometry. CUBIT supports sweeping options to convert imported FASTQ geometries into volumetric regions.

Importing STEP Files

The ACIS STEP translator provides bi-directional functionality for data translation between ACIS and the file format standard STEP AP203.

STEP AP203 is an international standard which defines a neutral file format for representation of configuration control design data for a product.

Prior to importing a STEP file for the first time into CUBIT, the STEP toolpath must be set. See Setting up CUBIT to use STEP tools for a description of how to do this.

The command used to import a STEP file are:

Import Step '<step_filename>' [no_bodies][no_surfaces] [no_curves] [no_vertices] [HEAL|noheal] [logfile ['filename'] [display]] [Group {'<name>'|<id>}] [sort] [XML '<xml_filename>']

The following describes the options that can be used when importing a STEP file:

[no_bodies][no_surfaces] [no_curves][no_vertices]

It is possible to include free entities (vertices, curves and surfaces) in the file. The default operation is to read all entities in the file whether they are included as part of a body or are free. By using any of the options **no_bodies**, **no_surfaces**, **no_curves**, or **no_vertices**, the user may exclude certain types of free entities.

[HEAL|noheal]

As with ACIS file import, you can control which types of entities to read. By default, bodies are automatically healed when imported - if this causes problems, you can disable this option by using the noheal argument. Also, you can optionally request a detailed logfile of the conversion process and display it in a text editor.

[logfile ['filename']

Specify a filename where informational messages generated during import of the STEP file will be written.

[group]

The group option of the import command will allow the user to create a group for each set of imported geometry. The newly created group can later be accessed using the name or id specified with the group option.

[sort]

Normally the numerical IDs of the geometric entities contained in the STEP model are used directly within CUBIT. The sort option provides the capability to compress the IDs read from the STEP file. The sort option does the same thing as the compress ids sort command, but combines it with the import command to remove a step in the process.

[XML '<xml_filename>']

This option will read assembly information and other metadata from an XML file in the DART metadata XML format. See the metadata documentation and the Analyst's Home Page for details.

Exporting a STEP file from Pro/Engineer

To export a STEP file from Pro/ENGINEER, from the Export STEP Dialog, Press Options.

In the file step_config.pro add the following:

STEP_EXPORT_FORMAT AP203_CD.

Also be sure your export option is set to Solids. If the geometry has problems in CUBIT, you may need to increase the geometry accuracy in Pro/ENGINEER.

Setting Up CUBIT to Use STEP Tools

In order to use the STEP import and export functionality, Cubit needs to know where the STEP tools are. There are two ways to do this:

1) Set the environment variable CUBIT_STEP_PATH to the correct path.

The correct path will be the path in the ACIS directories which ends in something like:

step/tools/xxx

where **xxx** would be the type of machine being used. An example path would be (for a Compaq Alpha machine)

/usr/local/eng_sci/cubit/acis/acis6.2/step/tools/osf

2) At the "CUBIT>" prompt type:

set steptools 'path/to/tools'

Note that the STEP import and export functionality might not be available on all 64-bit platforms.

See also Exporting STEP Files.

Importing IGES Files

The ACIS IGES translator provides bi-directional functionality for data translation between ACIS and the IGES (Initial Graphics Exchange Specification) format.

The commands to import IGES files are:

Import Iges '<iges_filename>' [no_bodies] [no_surfaces] [no_curves] [no_vertices] [Group {'<name>'|<id>}] [nofreesurfaces] [logfile ['filename'] [display]] [sort]

The following describes the options that can be used when importing IGES files:

[no_bodies] [no_surfaces] [no_curves] [no_vertices]

It is possible to include free entities (vertices, curves and surfaces) in the file. Default operation is to read all entities in the file whether they are included as part of a body or are free. By using any of the options **no_bodies, no_surfaces, no_curves**, or **no_vertices**, the user may exclude certain types of *free* entities.

[group]

The group option of the import command will allow the user to create a group for each set of imported geometry. The newly created group can later be accessed using the name or id specified with the group option.

[nofreesurfaces]

The **nofreesurfaces** option will automatically convert free surfaces to bodies. By default this option is **off**.

[logfile ['filename']]

Specify a filename where informational messages generated during import of the IGES file will be written.

[sort]

Normally the numerical IDs of the geometric entities contained in the ACIS model are used directly within CUBIT. The sort option provides the capability to compress the IDs read from the ACIS file. The sort option does the same thing as the compress ids sort command, but combines it with the **import** command to remove a step in the process.

Manifold Solid B-rep Objects (MSBO)

This translator supports Manifold Solid B-rep Objects (MSBO) as well as Trimmed Surface Objects. By default, MSBO objects (i.e., bodies) will be converted to trimmed surfaces. If you want to support MSBO objects during import, use this command (the default is *off*):

set AcisOption Integer 'iges_proc_msbo' On

You can add this to your .cubit file so it is turned on during each session of CUBIT.

Note that the IGES import and export functionality might not be available on all 64-bit platforms.

See also Exporting IGES Files.

Importing Facet Files

CUBIT provides the capability to import a model composed of facets to create geometry. The command to import facets from a file is:

Import [Facets|AVS|STL] "<filename>" [Feature_Angle] [LINEAR||Spline] [MERGE|no_merge] [make_elements] [stitch] [improve]

Facets are simply triangles that have been stitched together to form surfaces. Facetted geometry representations are commonly used for graphics, bio-medical, geotechnical and many other applications that output a discrete surface representation. Upon import, the resulting geometry representation is Mesh-Based Geometry. Figure 1. shows an example of a facetted model and the resulting geometry created in CUBIT.

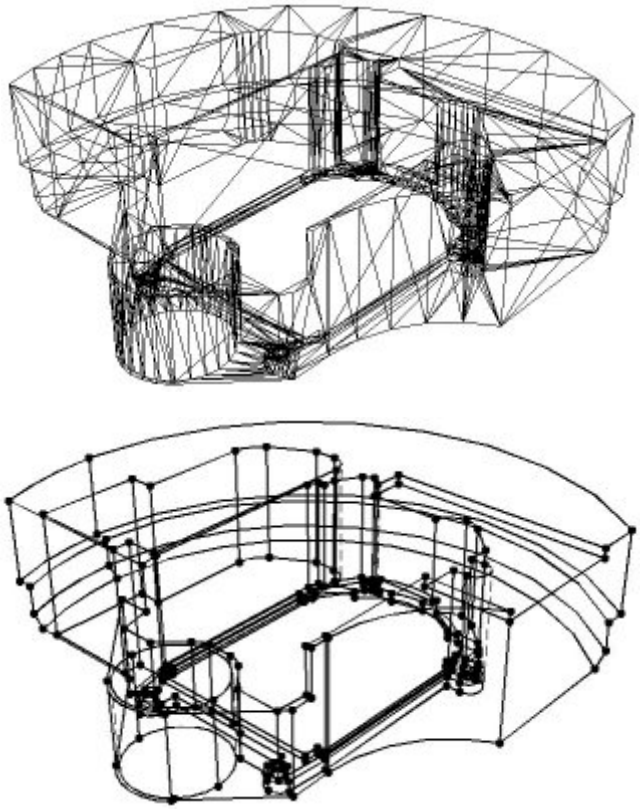


Figure 1. Example of facetted model and the resulting solid model created in CUBIT from the facets.

For convenience, the import facet command currently supports three different formats, facet, AVS and STL

- **Facet format**: The facet file format is a simple ASCII file that contains vertex coordinates and connectivities. The facet file format is described below.
- **AVS format**: The AVS format is a general geometry format that can support a variety of polygonal shapes. In CUBIT's implementation of the AVS import, it will support only triangles.
- **STL format**: Perhaps the most common format in the industry is Stereolithography (STL). CUBIT supports both ASCII and binary forms of the STL format. While the STL format is adequate for graphics and visualization, it can be problematic for geometry applications such as CUBIT. Each triangle in the STL format is represented independently. This means that multiple definitions of a single vertex are included in the file. CUBIT will attempt to merge duplicate vertices to form a water-tight surface. In cases where the vertex locations may not correspond exactly, an optional **tolerance** argument may be used on the import command. The **tolerance** option is used only for STL format files.

Facet File Format

The format for the ASCII facet file is as follows

```
n m
id1 x1 y1 z1
id2 x2 y2 z2
id3 x3 y3 z3
.
.
.
idn xn yn zn
fid1 id<1> id<2> id<3> [id<4>]
fid2 id<1> id<2> id<3> [id<4>]
fid3 id<1> id<2> id<3> [id<4>]
.
.
.
fidm id<1> id<2> id<3> [id<4>]
```

Where:

```
n = number of vertices
m = number of facet
id<i> = vertex ID if vertex i
x<i> y<i> z<i> = location of vertex i
fid<j> = facet ID if facet j
id<1> id<2> id<3> = IDs of facet vertices
[id<4>] = optional fourth vertex for quads
```

As noted above, the facets can be either quadrilaterals or triangles. Upon import, the facets serve as the underlying representation for the geometry. By default, the facets are not visible once the geometry has been imported. To view the facets, use the following command:

```
draw surf <id range> facets
```

Feature Angle

The **feature angle** option is used to specify the angle at which surfaces will be split by a curve or where curves will be split by a vertex. 180 degrees will generate a surface for every facet, while 0 degrees will define a single, unbroken surface from the shell of the mesh. The default angle is 135 degrees. This feature is identical to the feature angle option available when importing Exodus II files.

Smooth Curves and Surfaces

This option permits the use of a higher order approximation of the surface when remeshing/refining the resulting geometry. Default is to use the original facets themselves as the curve and surface geometry representation. If the facet model to be imported is to represent geometry with curved surfaces, it may be useful to apply this option. If the Spline option is selected, it will use a 4th order B-Spline approximation to the surface [Walton,96]. More information on using smooth approximation of the facets is available in Importing an Exodus II File.

Merge

This option allows the user to either merge or not merge the resulting surfaces. The default option is to merge adjacent surfaces. This results in non-manifold topology, where neighboring surfaces share common curves. The **no_merge** option, adjacent surfaces will generate distinct/separate curves.

Make elements

This option creates mesh elements from each of the facets on the facet surface.

Stitch

The stitch option is used with the facet or avs format files to try to merge vertices and triangles that are close. Figure 2 shows an example of where this might be employed. The model on the left contains facets that are not connected between the red and blue groups. In this case, the surfaces will not be water-tight, even though the vertices on the boundary between the two groups may be coincident. The **stitch** option attempts to eliminate the extra edge and vertex between the groups to form the model on the right. This option can be useful when importing facet files for 3D meshing. CUBIT's 3D meshing algorithms require a water-tight (closed) set of surfaces.

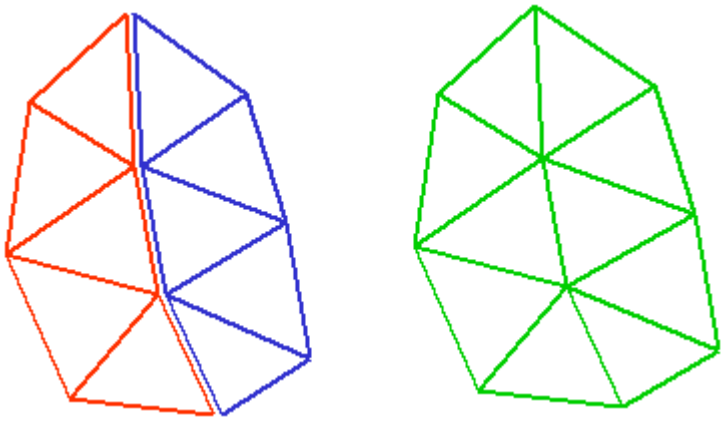
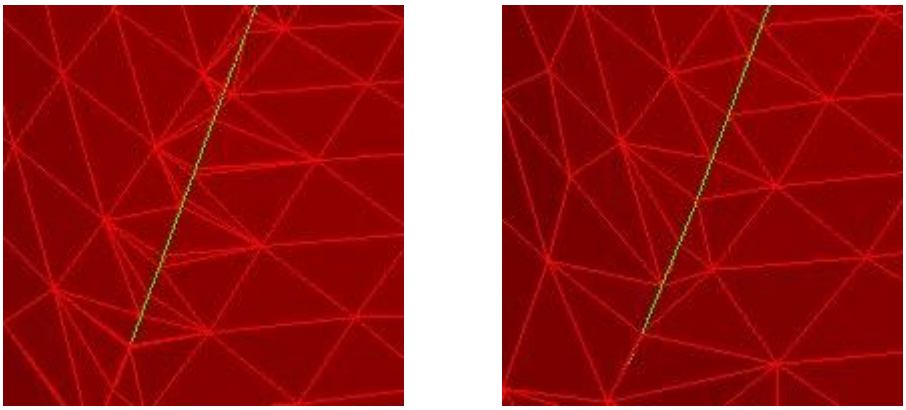


Figure 2. Example use of the stitch option on import.

Improve

The **improve** option will collapse short edges on the boundary of the triangulation that are less than 30% the length of the average edge length in the model. In some cases, short edges are the result of discrete boolean operations on the triangulation which may result in edges that are of negligible length. This option is particularly useful for boundaries where multiple surfaces come together at an edge. Figure 3. shows an example of where the improve option improved the quality of the triangles at the boundary. This option is especially useful if the facets themselves will be used for the FEA mesh.



Triangles near a boundary that have not been used the improve option | **The same set of triangles where improve option has collapsed edges**

Figure 3. Example use of the improve option

Importing Granite Files

Granite files consist of granite models with the (*.g) file extension. Granite models may be imported directly into CUBIT using the following command.

Import Granite '<granite_filename>'

When importing a granite file, the "set geometry engine granite" command will automatically be issued to set the appropriate geometry kernel.

The Granite kernel can also import the following geometry types:

- Pro/E part files (*.prt)
- Pro/E assembly files (*.asm)
- IGES files
- STEP files
- Granite files exported from cubit
- Granite Neutral files (not tested yet)

Exporting Geometry

Geometry can be exported from CUBIT in a variety of formats, including the ACIS ".sat" and ".sab" formats as well as in more portable exchange formats like STEP and IGES.

- Exporting ACIS Files
- Exporting STEP Files
- Exporting IGES Files
- Exporting Granite Files
- Exporting Facet Files

Exporting ACIS Files

Geometry can be exported from within CUBIT to the ACIS "sat" (ASCII) and "sab" (binary) formats. These formats can be used to exchange geometry between ACIS-compliant applications. The command used to export geometry is:

Export Acis [Debug] 'filename' [<geometry_entity_list>] [binary|ascii] [current] [overwrite]

The filename should be enclosed in single or double quotes. By convention, binary and ASCII ACIS files use the .sab and .sat filename extensions, respectively. If a geometry entity list is not specified, the entire ACIS model is exported. A geometry entity list is specified in the same format used for other CUBIT commands (See Entity Specification). Note that the model is saved as manifold geometry, and will have that representation when imported back into CUBIT (See Non-Manifold Topology and Geometry Merging.)

When exporting, the filename extension will determine the default file type, either ASCII or binary. A .sat extension will default to ASCII; a .sab extension will default to binary. If you use a different file extension you can specify the type with the **[binary|ascii]** option (with an unsupported extension exporting will default to ASCII but importing requires the type to be specified). Binary files can be significantly faster but are not guaranteed to be upward compatible nor cross-platform compatible (although testing has determined compatibility between NT and HP/UX).

In the GUI version, the **current** option will set the default filename for autosave (cntrl-S or File->Save (auto inc)) to the imported filename. Also, the filename is then set in the window titlebar.

When exporting with the "**file overwrite**" option on, the software will check to see if the file exists already, and if it does, exporting will fail in the command line version or ask to confirm the overwrite in the GUI version of CUBIT. The **overwrite** option will override this option and overwrite the file. The "file overwrite" option defaults to ON in the GUI version, OFF in the command line version.

When exporting, you can set the version of the Acis geometry. This allows backwards compatibility to previous versions of Cubit or other Acis-based applications. The command to change the Acis geometry engine version is: **Set Geometry Version [version_number]**
where **version_number** can be one of the following: 106, 107, 201, 300, 301, 401, 402, 403, 500, 501, 502, 503, 600, 601, 602, 603, 700 Note that you cannot set a version number that is higher than that of your current engine. For example, Cubit 6.0 was based on Acis 6.2, so you cannot set a geometry version of 700.

See also Importing ACIS Models.

Exporting STEP Files

CUBIT can export geometry to the STEP format, an emerging standard for storing geometry and other information. The STEP AP203 and STEP AP214 standards are supported. It is recommended to use AP214 for exchange of geometry information with CUBIT. The command used to export a STEP file is:

Export Step 'filename' [<geometry_entity_list>] [logfile ['filename'] [display]] [overwrite]

As with ACIS file export, you can specify which individual entities to export. If unspecified, all ACIS entities are exported.

The **logfile** option is used to save information regarding the conversion to STEP format. This information saved to a file with the name specified by the user, or named 'step_export.log' by default. When running the GUI version of CUBIT, the logfile can be displayed in a dialog window by using the **display** option.

The overwrite option works the same as with ACIS file export.

See Importing STEP Files for information on setting up the STEP import and export functionality.

Exporting IGES Files

The ACIS IGES translator provides bi-directional functionality for data translation between ACIS and the IGES (Initial Graphic Exchange Standard) format. The command to export IGES files is:

Export Iges 'filename' [<geometry_entity_list>] [logfile ['filename'] [display]] [overwrite]

As with ACIS file export, you can specify which individual entities to export. If unspecified, all ACIS entities are exported.

The **logfile** option is used to save information regarding the conversion to IGES format. This information saved to a file with the name specified by the user, or named 'iges_export.log' by default. When running the GUI version of CUBIT, the logfile can be displayed in a dialog window by using the **display** option.

The overwrite option works the same as with ACIS file export.

See also Importing IGES Files

See Importing IGES Files for information on setting up the IGES import and export functionality.

Exporting Granite Files

Granite files may be exported from CUBIT using the following command:

Export Granite '<granite_filename>' [Body <id_list>] [Volume <id_list>] [Surface <id_list>] [Curve <id_list>] [Vertex <id_list>]

The following formats can also be exported from Granite formats.

- IGES files
- STEP files
- ACIS SAT files. Note: The ACIS kernel cannot export Granite files.
- Granite files. Note: These files can only be read into CUBIT. Pro/E cannot read these files.

Exporting Facet Files

Facet files may be exported directly, or by converting from an ACIS or Granite representation. The syntax for exporting facet files is:

Export Facets 'filename' <entity_list> [overwrite]

The overwrite function allows you to overwrite an existing facet file.

Importing a Mesh

- Importing a Free Mesh
- Importing 2D Exodus II Files
- Importing Exodus II Files

ExodusII finite element data files can be imported into CUBIT. Several options for importing the mesh are available, (including mesh transformations):

- Importing a free mesh without geometry.
- Importing a free mesh and associating the mesh with ACIS-based geometry currently residing in CUBIT.
- Importing a 2D mesh and constructing ACIS-based Geometry
- Importing a mesh and constructing Mesh-Based Geometry from dihedral angles and boundary conditions.

The first two options listed above are described in detail in Importing a Free Mesh. The third option, discussed in Importing 2D Exodus II Files, is a limited capability for constructing ACIS geometry from a 2D mesh. The fourth option of creating Mesh-Based Geometry [Owen.2001] from the nodes, elements and boundary conditions of a finite element mesh is described in Importing an Exodus II File.

Importing 2D Exodus Files

CUBIT has a limited capability to create ACIS Geometry from 2D ExodusII finite element mesh files. (For a more general capability, see the Import Mesh Geometry command, which will create Mesh-Based Geometry).

To import a 2D Exodus II file and create ACIS geometry, the following command can be used:

import free mesh '<filename>' {time <t> | step <step#> | last}

CUBIT can create ACIS geometry from 2D Exodus II data files (4, 8, or 9 node QUAD or SHELL element types) that do not have enclosed voids (holes surrounded by mesh) and which were originally generated with CUBIT and exported to ExodusII with the Nodeset Associativity option set to on. The Nodeset Associativity command records the topology of the geometry into special nodesets which allow CUBIT to reconstruct a new solid model from the mesh even after it has been deformed. The new solid model of the deformed geometry can be remeshed with standard techniques or meshed with a sizing function that can also be imported into CUBIT from the same ExodusII file. CUBIT's implementation of the paving and triadvance algorithms can generate a mesh following a sizing function to capture a gradient of any variable (element or nodal) present in the ExodusII file.

In order for this feature to be effective, the following commands must be issued when the mesh is exported and later imported:

nodeset associativity on

set associativity complete on

The first command ensures that the geometry will be correctly recovered from the mesh, while the second ensures that boundary condition and material IDs will be recovered.

Importing Exodus II Files

Mesh-Based Geometry

CUBIT's mesh generation tools require an underlying geometry representation. In most cases, the ACIS solid modeling engine, compiled with CUBIT, is used to represent the geometry. However, in some cases, an ACIS representation is not available, and a previously developed finite element mesh is the only available representation of the model. In order to utilize CUBIT's mesh generation tools, the **import mesh geometry** command provides an option for creating geometry directly from the finite element mesh.

The **import mesh geometry** command will create a new volume for every block defined in the Exodus II file. It will also create curves, surfaces and vertices at appropriate locations on the model based on dihedral angles (also called feature angles) and assigned nodesets and/or sidesets. The mesh used to construct the geometry will be owned by the new geometric entities. This means that the mesh can be deleted, remeshed, or smoothed using any of CUBIT's meshing tools by simply using the new geometry definition. CUBIT will assign appropriate intervals to the new curves as well as determine an acceptable meshing scheme for surfaces and volumes.

The command to import a finite element mesh from an ExodusII format file and generate geometry from the mesh is:

Import Mesh Geometry '<exodusII_filename>'
[Block <id_range>|ALL]
[Use [NODESET|no_nodeset]
[SIDESET|no_sideset] [Feature_Angle <angle>] [LINEAR|Gradient|Quadratic|Spline]
[Deformed {Time <time>|Step <step>|Last} [Scale <value>]]
[MERGE|No_Merge]
[export_facets <1|2|3>]
[merge_nodes <tolerance>]

File Name

Type the name of file to import in single quotation marks. The file must reside in the current directory. For information on changing the current directory, see CUBIT environment commands . To list all the files in the current directory, type **ls** at the command prompt.

Blocks

Use this option to select the specific blocks to be imported from the Exodus II file. If no blocks are entered, then all blocks will be read and imported from the file. Standard ID parsing can also be used in this argument to select a range of blocks. For example "**1 to 5**" or "**1, 5 to 10 except 6**".

Each unique block selected to be imported will define a new body in the geometric model. Figure 1 shows a simple example of the geometry generated from the 3D finite element mesh.

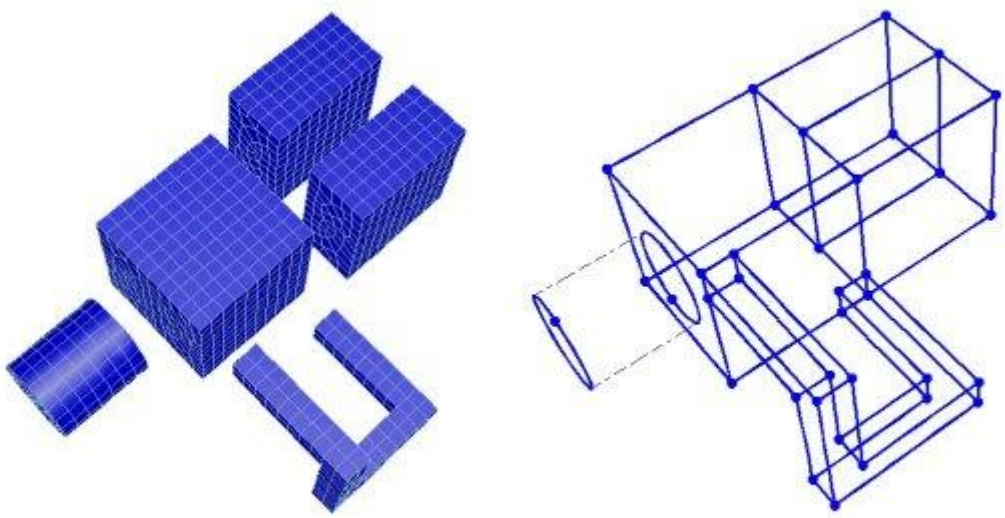


Figure 1. Example of mesh based geometry (right) created from a finite element mesh (left)

Blocks may be composed of 1D, 2D or 3D elements. For blocks composed of 2D elements (ie QUAD4, SHELL etc.), a sheet body will be created. One dimensional elements (ie. BEAM, TRUSS, etc.) will define curves. Where a block may be composed of more than one disconnected sets of elements, one body will be created for each continuous region of elements assigned to the same block. Where possible, the ID of the new body will be the same as the block ID. Since IDs must be unique, if a body ID is already in use, the next available ID will automatically assigned by the program.

Nodesets/Sidesets

Use the **nodeset** and **sideset** options to use any nodeset and sideset information in the Exodus II file in constructing geometry. Recall that nodesets and sidesets are generic boundary condition data assigned to nodes, edges or faces of the finite elements. It is useful to group mesh entities belonging to unique boundary conditions into geometric entities. This permits the user to remesh a particular region of the model without having to reassign boundary conditions.

If the **nodeset** and **sideset** arguments are given, geometric entities will be generated for each unique set of nodes, edges or element faces assigned to a nodeset or sideset. The default is to use any nodeset and sideset information available in the file. Figure 2 shows an example of how nodeset and sideset information might be used to generate geometry.

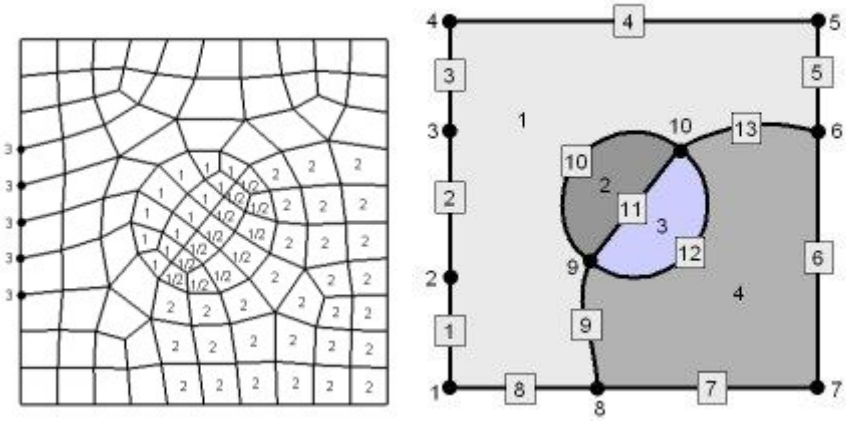


Figure 2. Example of geometry created from mesh entities assigned to nodesets (3) and sidesets (1 and 2).

Upon import, nodesets and sidesets are automatically created with the appropriate geometric entities assigned to them. The IDs of the new geometric entities, if generated from boundary condition data, will be the same as the nodeset and sideset IDs. Where doing so would conflict with existing geometric IDs, the program will automatically select the next available ID.

Feature Angle

Use this option to specify the angle at which surfaces will be split by a curve or where curves will be split by a vertex. 180 degrees will generate a surface for every element face, while 0 degrees will define a single, unbroken surface from the shell of the mesh. The default angle is 135 degrees.

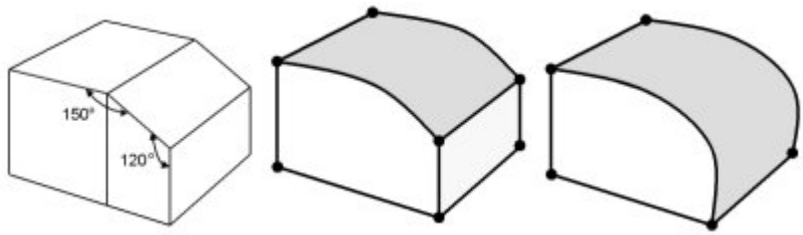


Figure 3. Example use of Feature Angle

Figure 3 shows an example of the use of different feature angles. On the left is a simple two-element hex mesh. Specifying a feature angle greater than 120 degrees would create the geometry in the center image. Using a feature angle less than 120 degrees and greater than 90 degrees would define the geometry on the right.

Smooth Curves and Surfaces

This argument allows the option of using a higher-order approximation of the surface when remeshing/refining the resulting geometry. Default is to use the original mesh faces themselves as the curve and surface geometry representation. If the finite element model to be imported is to represent geometry with curved surfaces, it may be useful to select this option. If selected, it will use a 4th order B-Spline approximation to the surface [Walton,96]. Figure 4 shows the effect of the smooth curve and surface option.

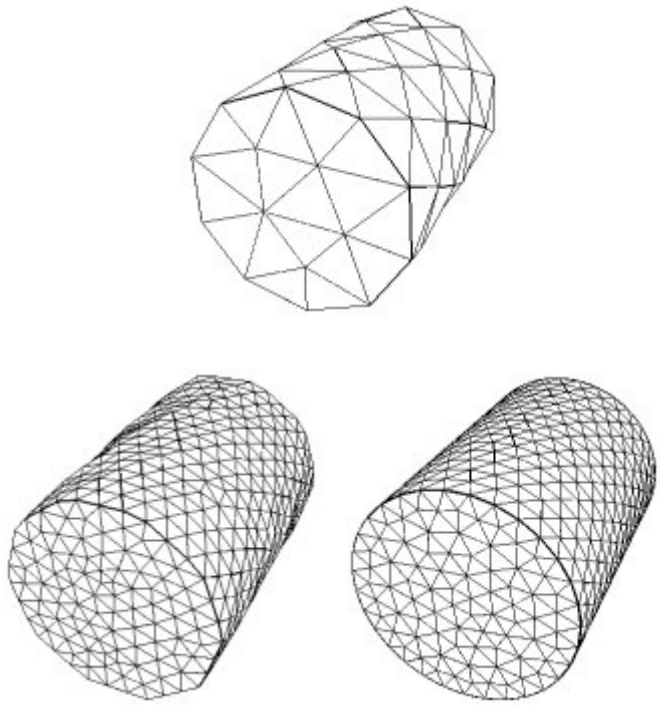


Figure 4. Effect of Smooth Curve and Surface Option for remeshing of mesh-based geometry

In this figure the top image is the original finite element mesh imported into CUBIT. In this example both models have been remeshed with the same element size. The difference is that the figure on the right use the smooth curve and surface option. While this option can improve the surface representation, it should be noted that memory requirements and meshing times can sometimes be affected.

If importing the Exodus II file using the command line, other options for surface representations are also available.

[LINEAR|Gradient|Quadratic|Spline]

The method used from the GUI is either **Linear** or **Spline**. The **Gradient** and **Quadratic** methods are still somewhat experimental and may not be as general purpose as the **Spline** representation.

Apply Deformations

This option permits the user to import time-dependant deformation information from the Exodus file. For this option, any vector data in the Exodus II file is assumed to be deformation information. If selected, deformations will be applied to the nodes upon import. Enter a specific time step value, integer step, or the last time available in the file. If time-dependant data is available in the Exodus II file, selecting the down arrow in the edit field will display the available time steps in the file. Default time is the last time step.

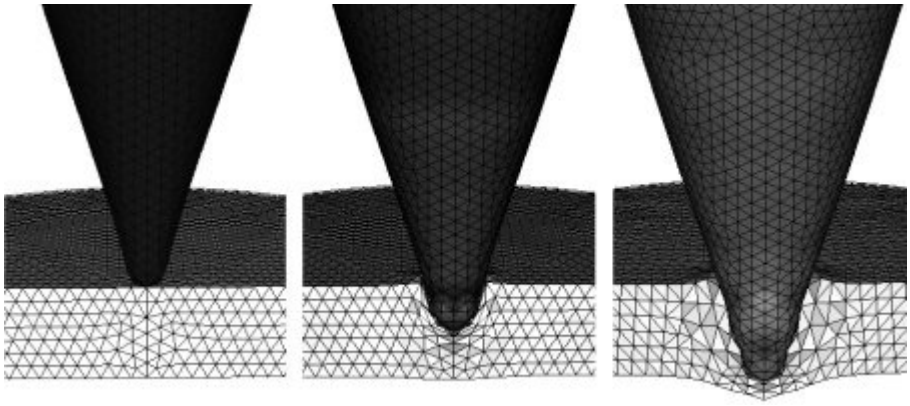


Figure 5. Example of remeshing of a deformed finite element mesh

Figure 5 shows an example of using Mesh-Based Geometry for a large deformation analysis. In this case, the analysis [Attaway et. al.,98] began and continued until mesh quality became unacceptable. At that point, the mesh was imported into CUBIT and geometry re-created from the computed deformations. The finite element mesh could then be removed, remeshed or improved and written back to an Exodus II file. After remapping [Wellman,99] the appropriate analysis variables back to the mesh, the analysis could then be restarted. This process was repeated multiple times until the desired results were achieved.

Note: Care should be taken when using large deformations, as inverted elements (negative Jacobians) may produce unpredictable results with the resulting geometric representation.

Also available is an optional **scale** factor. This applies the indicated scale to all deformations. Default is 1.0.

Merge

This option allows the user to either merge or not merge the resulting volumes. The default option is to merge adjacent volumes. This results in non-manifold topology, where neighboring volumes share common surfaces. Using the no_merge option, adjacent volumes will generate distinct/separate surfaces.

Merge Nodes

The **merge_nodes** option will allow the user to specify a different tolerance for merging nodes on import. The default value is 1e-6.

Note: Care should be taken when setting import merge tolerances. Setting a tolerance too low will not merge adjacent nodes. Setting the tolerance too high can produce undesirable results, and severely tangle the mesh.

Export Facets

[export_facets <1|2|3>]

This is primarily a debug option available only from the command line. This option will export the shell of the Exodus mesh to an ASCII file in the form of facets. The resulting file can be imported to Cubit using the "Import Facets" command. Export options: 1 = export only the exterior facets to file "facets.shell"; 2 = export only the interior facets between element blocks to file "facets.inter"; 3 = export all boundary facets to file "facets.all".

Importing a Free Mesh

The command to import a mesh from an Exodus II format file is:

> Import Mesh '<exodusII_filename>' [Block <block_ids>] [Unique Genesis IDs] [Shell]
> [{Group|Body|Volume|Surface|Curve|Vertex} <id_range> | Preview]

This command permits the user to import a mesh for visualization purpose or to import the mesh onto an existing geometry. CUBIT also has the capability to generate geometry directly from the nodes and elements of a finite element mesh. See also Importing Exodus II Files in this section of this manual. Related Commands:

Import Mesh Geometry (options)

Import Free Mesh (2D)

Delete Mesh Preview

Export [Genesis | Mesh] '<filename>'

List Import Mesh NodeSet Associativity

List [Export Mesh] NodeSet Associativity

[set] Import Mesh NodeSet Associativity [ON|off]

[set] [Export Mesh] NodeSet Associativity [on|OFF]

Transforming Mesh Coordinates

The user can import a mesh from an Exodus II file and associate the mesh with matching geometry. The resulting mesh may then be manipulated normally. For example, the mesh may be smoothed or portions of it deleted and remeshed. The user can save their work by exporting the geometry and mesh, and then restore the geometry and mesh later. In some cases, saving and restoring can be faster or more reliable than replaying journal files.

Saving and importing a mesh may be useful for teams working on creating a conforming mesh of a large assembly so that they can pass information to one another. For example, a team member can export the mesh on the surfaces between two parts, and another team member import the mesh for use on an adjoining part of the assembly.

As of cubit version 7.0, any higher order elements, block definitions, nodesets, and sidesets are retained on import.

Importing a Mesh with Nodeset Associativity

Meshes can be imported into CUBIT that contain nodeset associativity data used for defining finite element boundary conditions. If an exported CUBIT mesh is going to be imported back onto the same geometry, then before exporting the user should issue the following command:

> set export mesh nodeset associativity on

This causes extra nodeset data to be written, which associates every node to a geometric entity, resulting in an import which is more reliable. When importing, if the user does not want to use the nodeset associativity data that exists in a file, then before importing the following command should be used:

> set import mesh nodeset associativity off

The user may wish to turn geometry associativity off if, for example, the geometry is no longer identical as a result of curves being composited, or CUBIT names changed due to a ACIS version changes.

Importing a Mesh onto Modified Geometry

Although there are some exceptions, CUBIT requires that the mesh be imported onto the same geometry from which it was exported.

Since merge information is not stored with the ACIS representation, care should be taken that the geometry is merged the same way on export and import of the mesh. If not, importing the mesh one block at a time in successive commands may increase the chance of a successful import, at the cost of more memory and time.

Between exporting and importing a mesh, the geometry may be modified slightly by compositing entities. Mesh import will, however not be successful if entities are partitioned or a body is webcut. In some cases mesh import may be successful on modified geometry if the new vertices match up exactly with nodes of the mesh, and the new curves match up exactly with edge chains of the mesh. Unless this criteria is met, associating the mesh with the geometry will be unsuccessful.

Importing a Mesh without Geometry Associativity

A mesh may be imported without associating the nodes and elements to geometry by using the **Preview** option. This may be useful, if importing the mesh is unsuccessful with the current geometry representation. In most cases this option is used only to preview the mesh in order to determine where geometry associatively problems may exist. Support for meshes without geometry associativity is limited to **List**, **Draw** and view navigation commands.

When a mesh is imported with the **Preview** option, the imported mesh entities are placed in a group called **free_elements**. To see if the elements match the geometry, the user may issue the following command:

```
draw free_elements add
```

To delete the unassociated mesh elements, use the following command:

```
delete mesh preview
```

Specifying a Portion of the Mesh to be Imported

The **Block** option in the **Import Mesh** command indicates that only the specified element block should be imported from the Exodus II file. In the same manner, the **Volume** and other geometry options provide a way to import the nodes and element on the indicated geometry. If neither a **block** nor a **geometry entity** is specified, then the entire mesh file is read.

If a **block** is specified without specifying a **geometry entity**, associativity or proximity is used to determine which volume the block elements should be associated with. If a **block** and a **volume** are specified, the block elements are associated with the specified volume, provided they actually match. If a **volume** is specified without a **block**, associativity data is used to find a block corresponding to the given volume.

Unique Genesis IDs and Shell Options

The Unique Genesis IDs option is used to preserve ids in the genesis file in the case that id overlap exists when importing into CUBIT. This can occur when importing into an active session where CUBIT ids have already been assigned.

The Shell Option is used as a flag to alert the program that there are shell elements in the file. Shell elements can not always be detected by the import program, and this ensures that the shell elements will be included in the model.

Exporting the Finite Element Model

CUBIT currently supports the Exodus file format for exporting the finite element model.

- Exporting an Exodus II File
- Exporting an ABAQUS File

Other Formats

Cubit also has limited export capabilities for Nastran, Patran and Ideas mesh files.

Export Ideas '<filename>' [Node <id_list> Hex <id_list> Tet <id_list//> Face <id_list>] Tri <id_list>] [overwrite]
Export Patran '<filename>' [overwrite]
Export Nastran '<filename>' [overwrite]

Note that only the mesh information will be exported. CUBIT doesn't support boundary conditions exported from Cubit in these formats.

Custom translators are available to translate between the Exodus II format and a limited number of other analysis code formats. Contact the cubit development team for a current list of supported translator formats.

Exporting an Exodus II File

After defining the element blocks, nodesets and sidesets for a model, the model can be written to the Exodus II file using the command:

Export [Genesis|Mesh] '<filename>' [dimension {2|3}] [Block <id_list>] [XML '<filename>']

The **Genesis** or **Mesh** arguments are optional and both indicate that an Exodus II format will be written. The filename can be any valid filename. Where a full path is not specified, the file will be written in the current working directory.

The **dimension** argument is also optional. Most element types have an inherent dimensionality associated with them. For example, a truss or beam element is inherently 2D while a hex or tetra element is 3D. Without this argument, only the x-y location of the nodal coordinates of 2D elements are written to the Exodus II file. Using the argument dimension 3, in this example, permits the full 3D coordinates to be written.

The optional **Block** argument may also be added to the **Export** command. Without this argument, all blocks defined in the current model will be exported to the Exodus II file. This argument permits the user to specify only a portion of the blocks in the model. The **<id_list>** may be any valid set of integers corresponding to the Blocks in the current model.

The **XML** optional argument may also be added to the **Export** command. When this argument is included and assembly data exists in the model, an XML file is written which describes the relationship between block IDs in the Exodus II file and parts in the assembly. See the Metadata section for details.

Converting an Exodus II file to ASCII

The Exodus II file format is binary. It is frequently necessary to view the contents of the Exodus II file as plain text. A publicly available tool known as **ncdump** can be used to view the contents of an Exodus II file. **ncdump** is part of the **netCDF** library and is currently available from Unidata at the following URL:

http://www.unidata.ucar.edu/

On a UNIX platform, typical use of the **ncdump** utility is:

ncdump filename.e > filename.txt

In this format, the **ncdump** utility will take the Exodus II file, **filename.e**, and dump the contents to an ASCII file **filename.txt**

Another option for converting between binary and ASCII formats of Exodus II files is a utility known as **exotxt**. Exotxt is part of the SEACAS tool suite. Contact the Sandia CUBIT development team for a copy of this utility.

Note that the 'stock' ncdump utility should work for most meshes; however, Sandia increases some of the dimensions in order to handle larger meshes (more element blocks, boundary conditions, variables). The dimensions we increase in netcdf.h are:

NC_MAX_DIMS (max dimensions per file) from 100 to 65536
NC_MAX_VARS (max variables per file) from 2000 to 524288

Exporting ABAQUS Files

To export a mesh to an ABAQUS file, issue the command:

Export Abaqus '<filename>' [Block <id_list>]

The command is nearly identical to the stand-alone utility, exoaba.

The ABAQUS export command only supports a subset of CUBIT's element types, namely HEX, HEX8, QUAD,QUAD4, SHELL, SHELL4, TETRA, TETRA4, TRI, TRI3, BAR, BAR2, BAR3, TRUSS, TRUSS2, TRUSS3, BEAM, BEAM2, BEAM3, and SPRING. As with exodus and exoaba, blocks, nodesets, and sidesets are exported as generic boundary conditions, not as specific types of boundary conditions (like a point load).

Block attributes and BC distribution factors are not exported to the abaqus file.

Finite Element Model

- Finite Element Model Definition
- Element Block Specification
- Nodesets and Sidesets
- Exodus II Model Title
- Transforming Mesh Coordinates
- Exodus Coordinate Frames
- Exodus II File Specification

This chapter describes the techniques used to complete the definition of the finite element model. The definitions of the basic items in an Exodus database are briefly presented, followed by a description of the commands a user would typically enter to produce a customized finite element problem description. Commands for exporting the finite element model are given in the Importing and Exporting Data chapter.

Finite Element Model Definitions

Sandia's finite element analysis codes have been written to transfer mesh definition data in the ExodusII file format (citation Schoof, 95). The ExodusII database exported during a CUBIT session is sometimes referred to as a Genesis database file; this term is used to refer to a subset of an Exodus file containing the problem definition only, i.e., no analysis results are included in the database.

The ExodusII database contains mechanisms for grouping elements into Element Blocks, which are used to define material types of elements. ExodusII also allows the definition of groups of nodes and element sides in Nodesets and Sidesets, respectively; these are useful for defining boundary and initial conditions. Using Element Blocks, Nodesets and Sidesets allows the grouping of elements, nodes and sides for use in defining boundary conditions, without storing analysis code-specific boundary condition types. This allows CUBIT to generate meshes for many different types of finite element codes.

Element Blocks

Element Blocks (also referred to as simply, Blocks) are a logical grouping of elements all having the same basic geometry and number of nodes. All elements within an Element Block are required to have the same element type. Access to an Element Block is accomplished through a user-specified integer Block ID. Typically, Element Blocks are used by analysis codes to associate material properties and/or body forces with a group of elements.

Nodesets

Nodesets are a logical grouping of nodes accessed through a user-specified Nodeset ID. Nodesets provide a means to reference a group of nodes with a single ID. They are typically used to specify load or boundary conditions on portions of the CUBIT model or to identify a group of nodes for a special output request in the finite element analysis code.

Sidesets

Sidesets are another mechanism by which constraints may be applied to the model. Sidesets represent a grouping of element sides and are also referenced using an integer Sideset ID. They are typically used in situations where a constraint must be associated with element sides to satisfactorily represent the physics (for example, a contact surface or a pressure.

Element Types

The basic elements used to discretize geometry were described in the mesh generation chapter. Within each basic element type, several specific element types are available. These specific element types vary by the number of nodes used to define the element, and result in different orders of accuracy of the element. The element types available for each basic element type defined in CUBIT are summarized in the following table. For a description of the node and side numbering conventions for each specific element type, see the Appendix. Element types can be set for individual Element Blocks, either before or after meshing has been performed. Higher-order nodes are created only when the mesh is being exported to the Exodus II file, and persist in the CUBIT database after file export.

Table 1. Element Types Defined in CUBIT

Basic Element Type	Specific Element Type	Notes
Edge	BAR, BEAM	Bars have 2 DOF's per node, Beams 3
Triangle	TRI, TRI3, TRI6, TRI7, TRISHELL, TRISHELL3, TRISHELL6, TRISHELL7	Tri element nodal coordinates are always 3D.
Quadrilateral	QUAD, QUAD4, QUAD8, QUAD9; SHELL, SHELL4, SHELL8, SHELL9	Quad element nodal coordinates are 2D, that is their nodes contain only x and y coordinates. Shell element nodal coordinates are 3D.
Tetrahedron	TETRA, TETRA4, TETRA8, TETRA10	TETRA8 contains vertex nodes and mid-face nodes, experimental element used in Sandia FEA research
Hexahedron	HEX, HEX8, HEX20, HEX27	

Element Block Specification

- Creating Blocks
- Default Element Blocks
- Defining the Element Type
- Assigning Attributes
- Displaying Blocks
- Deleting Blocks
- Automatically Assigning Mesh Edges to a Block (Rebar)
- 2d Elements

Element blocks are the method CUBIT uses to group related sets of elements into a single entity. Each element in an element block must have the same basic and specific element type.

The preferred method for defining blocks is to use geometric entities such as volumes, surfaces or curves. Blocks can also be defined using mesh entities. If a block is defined at a geometric entity, each of the elements *owned* by the geometry are automatically assigned to the block. Deleting or remeshing the geometry automatically changes the set of elements grouped into the block. If mesh entities are used to specify a block, deleting the mesh will also delete the elements from the block.

Some important notes regarding Element Blocks are as follows:

- Multiple volumes, surfaces, and curves can be contained in a single element block
- A volume, surface, or curve can only be in one element block

- Element Block id's are arbitrary and user-defined. They do not need to be in any contiguous sequence of integers.
- Element Blocks can be assigned a single floating point number, referred to as the block Attribute; this number is used to represent the length or thickness of Bar and Shell elements, respectively.The attribute defaults to 1.0 if not specified.

Creating Element Blocks

Element blocks are defined with the following Block commands.

Block <block_id> {Vertex | Curve | Surface | Volume} <range> [Remove]

Block <block_id> {Hex|Tet|Pyramid|Face|Tri|Edge|Node} <range> [Remove]

Block <block_id> Group <range> [Remove]

The first command defines the block based on a list of geometric entities, while the second uses specific lists of mesh entities. Since a block can only contain a single element type, usually entities of the same type are defined on the same block. The third option provides for assigning **groups** of entities to a single block. This is useful, for example, when several entities of the same type can be grouped together. The **Block Group** command simplifies the specification of the block.

By using the **Remove** argument to the **block** command, the specified geometry or mesh entity can be removed from the block definition.

Default Element Blocks

When exporting an ExodusII file, if the user has not specified any Element Blocks, by default element blocks will be written for any meshed volumes. This default behavior can be changed, to write surface, volume, or no meshes by default. This option can be set using the command

Set Default Block [ON|off|Volume|Surface]

Default behavior, **ON**, is for the blocks to automatically be written based on their owning geometry. When the **OFF** setting is used, only the mesh contained in blocks created by the user will be exported. Mesh not in an element block at export time, will not be exported. The export will still succeed and no error will be thrown. If **Volume** is specified, only elements contained in volumes will have default blocks specified. Similarly, the **Surface** arguments indicates that only surfaces containing elements will use default blocks.

When default blocks are used, the IDs for the resulting blocks will be defined as follows based upon the type of geometry:

Volume: The default block ID will be set to the Volume ID

Surface: The block ID will be set to 0

Curve: The block ID will be set to 0

Defining the Element Type

Each block must have a specific element type associated with it. To assign an element type to a block, use the following command:

Block <block_id_range> Element Type <type>

Available element types are defined by the Exodus II file format specification (Schoof, 95). CUBIT supports the following element types:

Nodes: SPHERE (there is no other choice of this type)

Curves: BAR BAR2 BAR3 BEAM BEAM2 BEAM3 TRUSS TRUSS2 TRUSS3

Surfaces: QUAD QUAD4 QUAD5 QUAD8 QUAD9 SHELL SHELL4 SHELL8 SHELL9 HEXSHELL
TRI TRI3 TRI6 TRI7 TRISHELL TRISHELL3 TRISHELL6 TRISHELL7

Volumes: HEX HEX8 HEX9 HEX20 HEX27 PYRAMID TETRA TETRA4 TETRA8 TETRA10 TETRA14

If the element type is not assigned for an element block, it will be assigned a default type depending on which type of geometry entity is contained in the block. The default values used for element type are:

Volume: 8-node hexahedral elements (HEX8) will be generated for hex meshes. TETRA4 will be generated for tet meshes.

Surface: 4-node shell elements (SHELL4) will be generated for quad meshes and TRISHELL3 for tri meshes.

Curve: 2-node bar elements (BAR2) will be generated.

Node: 1-node elements (SPHERE) will be generated.

Higher order nodes are moved to curved geometry by default. To change this, use the following command:

set Node Constraint [ON|off]

On means higher order nodes snap to curved geometry. **Off** means higher order nodes are placed at the average location of the element nodes: for edges, this means on the line containing the edge; for 2d elements, this usually means on the plane containing the element. Several examples of specifying various types of element blocks are given in the Appendix.

Assigning Attributes to Blocks

It may be necessary to associate attributes with a specific element block. Attributes are generally integer or floating point values that represent some physical property in the region occupied by the block, such as material properties or shell thickness. To assign an attribute to an element block, use the following command:

Block <block_id_range> Attribute <value>

The default number of attributes of an element block is dependent on the element type of the element block. Except for the element blocks of the element types below, all element blocks contain zero attributes by default.

Element Type	Number Default Attributes
SPHERE	1
BAR	1
BEAM	3
TRUSS	1
SPRING	1
SHELL	1
TRISHELL	1

To assign more attributes than the number of default attributes use the following command:

Block <id_range> Attribute Count <1-10>

CUBIT will store up to 10 attributes per block. Specify the maximum number of attributes to be stored on the block with this command. Once this command has been executed, individual attributes may be set using the following command:

Block <id_range> Attribute Index <index> <value>

The index is an integer from 1 to the maximum count specified in the Block Attribute Count command. The value may be any valid floating point number.

Displaying Element Blocks

Blocks can be viewed individually with CUBIT by employing the following command:

Draw Block <block_id_range> [Color <color_spec>] [add]

Block colors can also be changed using the following command:

Color Block <block_id_range> {color|Default}

Deleting Element Blocks

All Nodesets, Sidesets and Blocks may be deleted from the model using the following command:

Reset Genesis

To remove only Blocks, the following may be used:

Reset Block

To remove a specific block, use:

Delete Block <block_id_range>

Automatically Assigning Mesh Edges to a Block (Rebar)

After a mesh has been defined within a volume, it may be useful to use the existing mesh edges as the basis for an element block. Such an element block might be composed of bars or truss type elements that might propagate through a solid medium such as rebar placed in reinforced concrete. Although the **Block <id> Edge <range>** command could be used for this task, it would prove extremely tedious defining the individual edges to add to the block. To make this process easier, the following command can be used:

Rebar Start <x> <y> <z> Direction <x> <y> <z> [Length <value>] Block <id> [Element Type <type>]

The **Rebar** command allows the user to specify a starting location for a set of edges and an initial direction. The program will find the closest existing node in the mesh to **Start <x> <y> <z>** and begin propagating through the mesh in the specified **Direction <x> <y> <z>**, adding edges to the block as it propagates through the mesh. The edge that is attached to the last node and is within a fixed 30 degrees of the specified direction is added to the block. The Propagation of the edges continues until either the optional **Length** value is reached or an edge does meet the **Direction** criteria. Also required with this command is a **block** ID. An **Element Type** can also be specified, where the valid element types are any of the following:

Valid element types for Rebar command: BAR BAR2 BAR3 BEAM BEAM2 BEAM3 TRUSS TRUSS2 TRUSS3

A related command for creating curve geometry directly from mesh edges is the Create Curve from Mesh command. See Curve creation for more details.

2D Elements

CUBIT is a 3d mesh generator by default. Element types, by default, are respectively TRISHELL and SHELL for triangle and quad elements. If a 2d mesh is desired, blocks types must be explicitly set to TRI or QUAD.

Example:

create brick x 10
surface 1 scheme trimesh
mesh surface 1
block 1 surface 1
block 1 element type tri
export mesh "mymesh.exo"

Sideset 1 will be based on the TRI and QUAD elements in **blocks 1 and 2**, with the side numbering referring to the edges of the triangles and quads.

Nodeset and Sideset Specification

- Creating Nodesets and Sidesets
- Grouping Faces on a Surface into a Sideset
- Deleting Nodesets and Sidesets
- Displaying Nodesets and Sidesets
- Nodeset Associativity

Boundary conditions such as constraints and loads are applied to the finite element model using *nodesets or sidesets*, also known as Genesis entities. Rather than attempting to maintain specific boundary condition information, such as load, temperature, constraint, etc., Genesis entities are the generic vehicle for the user to set up boundary conditions on the model. Nodes, elements and element faces are instead grouped together and assigned unique IDs. Node, element and face IDs assigned to Genesis entities can then be written to the Exodus II mesh file. Once imported to the intended analysis application, the nodeset and sideset IDs can be appropriately interpreted as specific physical boundary conditions.

The preferred method for creating Genesis entities is to assign vertices, curves, surfaces or volumes to a specific nodeset or sideset ID. Any mesh entity *owned* by the geometric entity in a nodeset or sideset is automatically assigned to the same nodeset or sideset. This allows greatest flexibility in generating and updating the finite element mesh. For example, if a surface belongs to a specific sideset, remeshing the surface will automatically delete any old faces from the sideset and add the faces of the new mesh.

In some cases, the geometric model does not provide enough resolution to define the desired boundary conditions. In this case, the model may be partitioned using CUBIT's virtual geometry features. Where this may not be feasible, mesh entities can also be added directly to the desired nodeset or sideset. Where individual mesh entities have been added to nodesets or sidesets, deleting the mesh will also remove these elements from the Genesis entity. If the geometry is remeshed, the new mesh entities must also be added once again to the nodesets or sidesets.

Nodesets can be created from groups of nodes categorized by their owning volumes, surfaces, curves or vertex. Individual nodes may also be added to a nodeset. Nodes can belong to more than one nodeset.

Sidesets can be created from groups of element sides or faces categorized by their owning surfaces or curves or by their individual face IDs. Element sides and faces can also belong to more than one sideset.

Creating Nodesets and Sidesets

Nodesets and Sidesets are created in CUBIT by assigning the appropriate geometry or mesh entities in the model to a nodeset or sideset ID. The following commands can be used:

```
Nodeset <nodeset_id> {Curve | Surface | Volume | Vertex | Node} <range> [Remove]

Sideset <sideset_id> Group <id_range> [remove]

Sideset <sideset_id> {Curve|Surface|Edge|Face|Tri} <id_range> Remove

Sideset <sideset_id> Edge <id_range> [wrt {{Tri|Face} <id_range> | all } ]

Sideset <sideset_id> Face <id_range> [wrt {Hex <id_range> | all} ]

Sideset <sideset_id> Tri <id_range> [wrt {Tet <id_range> | all} ]

Sideset <sideset_id> Surface <id_range> [wrt {{Volume|Surface} <id_range> | all} ]
[FORWARD|Reverse|Both]

Sideset <sideset_id> Curve <id_range> [wrt {Surface <id_range> | all} ]
```

Like element blocks, Nodesets and Sidesets are given arbitrary, user-defined ID numbers. If there are no user-defined Nodesets or Sidesets, none are written to the Exodus II file.

With Sidesets, direction is often important. For surfaces, the direction may be specified using the **Forward**, **Reverse**, or **Both** options. The **Forward** option will write a sideset in relation to hexes in the surface's forward volume, which is the volume that the surface's normal points away from. The **Reverse** option will write a sideset in relation to hexes in the surface's reverse volume, which is the volume that the surface's normal points into. The **Both** option will allow sidesets to be written in relation to the hexes that lie in volumes on both sides of the surface. The default is **Forward**. The user can additionally specify the volume from which the hexes should be taken in relation to by using the **wrt Volume** option.

Direction is equally important for curves in Sidesets. The **wrt Surface** option allows the user to indicate which surface's faces will be included in the Sideset. The **wrt All** option will include all faces attached to the curve. The default is wrt All.

Grouping Faces on a Surface into a Sideset

A sideset can be created by grouping a portion of the faces on a given surface by using the following command.

SideSet <sideset_id> Surface <surf_id> Patch { Maximum <x> <y> <z> Minimum <x> <y> <z> |
Center <x> <y> <z> outer_radius <value> [inner_radius <value>]} [partition]

This command places only the faces meeting the specified criteria into the sideset. Specifying the **Maximum** and **Minimum** locations of a bounding box will place all faces on the **surface** whose centroid fall within the box defined by the **Maximum** and **Minimum** vectors. Using the Center and outer_radius option will place into the sideset, all faces on the surface whose centroids fall within the circle defined by **Center** and **outer_radius**. An optional **inner_radius** may also be specified, where faces within the annulus defined by the **inner_radius** and **outer_radius** are placed in the sideset. The **partition** option will split the surface based on the sideset definitions, creating new surfaces.

Important: Unlike the other Sideset commands that use geometric entities, this command does not assign the geometric surface to the sideset. Instead only the mesh entity faces are added. If the mesh is deleted, the sideset will become invalid.

Deleting Nodesets and Sidesets

All Nodesets, Sidesets and Blocks may be deleted from the model using the following command:

Reset Genesis

To remove only nodesets or sidesets, the following may be used:

Reset Nodeset

Reset Sideset

To remove a specific nodeset or sideset, use:

Delete Nodeset <nodeset_id_range>

Delete Sideset <sideset_id_range>

Displaying Nodesets and Sidesets

Nodesets and Sidesets can be viewed individually through CUBIT by employing the following commands:

Draw NodeSet <nodeset_id_range> [Color <color_spec>] [add]

Draw SideSet <sideset_id_range> [Color <color_spec>] [add]

Nodeset and Sideset colors can also be changed using the following commands:

Color NodeSet <nodeset_id_range> {color|Default}

Color SideSet <sideset_id_range> {color|Default}

Nodeset Associativity Data

Nodesets can be used to store geometry associativity data in the Exodus II file. This data can be used to associate the corresponding mesh to an existing geometry in a subsequent CUBIT session. This functionality can be used either to associate a previously-generated mesh with a geometry (See Importing an Exodus II File), or to associate a field function with a geometry for adaptive surface meshing (See Adaptive Meshing).

The commands to control and list whether associativity data is written or read from an Exodus II files are the following:

List Import Mesh NodeSet Associativity

List [Export Mesh] NodeSet Associativity

List [Export Mesh] NodeSet Associativity Complete

set Import Mesh NodeSet Associativity [ON|off]

[set] [Export Mesh] NodeSet Associativity [on|OFF]

[set] [Export Mesh] NodeSet Associativity Complete [On|OFF]

Associativity data is stored in the Exodus II file in two locations. First, a nodeset is written for each piece of geometry (vertices, curves, etc) containing the nodes owned for that geometry. Then, the name of each geometry entity is associated with the corresponding nodeset by writing a property name and designating the corresponding nodeset as having that property. Nodeset numbers used for associativity nodesets are determined by adding a fixed base number (depending on the order of the geometric entity) to the geometric entity id number. The base numbers for various orders of geometric entities are shown in the following table. For example, nodes owned by curve number 26 would be stored in associativity nodeset 40026.

Table 1. Nodeset ID base numbers for geometric entities

Geometric Entity	Base Nodeset ID
Vertex	50000
Curve	40000
Surface	30000
Volume	20000

Instead of storing just the nodes owned by a particular entity, nodes for lower order entities are also stored. For example, the associativity nodeset for a surface would contain all nodes owned by that surface as well as the nodes on the bounding curves and vertices.

Exodus II Model Title

CUBIT will automatically generate a default title for the Genesis database. The default title has the form:

cubit(genesis_filename): date: time

The title can be changed using the command:

Title '<title_string>'

Transforming Mesh Coordinates

A mesh can be scaled and transformed to a new location as it is written to or read from an Exodus file. To transform a mesh during import or export use the following command:

```
Transform Mesh {Input|Output}
[Scale <xyz_factor>]
[Scale <x_factor> <y_factor> <z_factor>]]
[Scale {X|Y|Z} <factor>]
[Translate <dx> [<dy> [<dz>]]]
[Translate {X|Y|Z} <distance>]
[Rotate <degrees> about {X|Y|Z}]
[Reset]
```

This command may be repeated any number of times using any number of options. Transform commands are cumulative, added to the effect of previous transforms. If more than one transformation is entered in the same command, transformations are applied in the order they appear in the command.

To clear a transformation matrix, use the **Reset** option:

```
Transform Mesh {Input|Output} Reset
```

Mesh input and output transformations are also cleared when you reset the entire model using the **Reset** command.

Transforming a mesh during output **does not** change the position of the mesh within CUBIT. It only changes the nodal positions written to the Exodus file. Nodal positions may be changed within CUBIT by transforming the body that contains the mesh. See Geometry Transforms for information on how to apply transformations to a Body.

Transforming a mesh during input **does** change the position of the mesh with CUBIT. The file being read is not modified.

Transformations applied during mesh input are independent of transformations applied during mesh output.

The following example generates a simple mesh, writes the mesh with its coordinates scaled by a factor of 2, and then re-imports that mesh, restoring the scaling to what it originally was in CUBIT.

```
brick x 10
volume 1 interval 4
mesh vol 1
transform mesh output scale 2
export mesh 'temp.exo'
delete mesh
transform mesh input scale .5
import mesh 'temp.exo'
```

See Geometry Transforms for information on how to apply transformations to a Body.

See Nodeset and Nodeset Repositioning

See Importing a Mesh

See Mesh Based Geometry

Exodus Coordinate Frames

CUBIT allows the user to define coordinate systems (frames) that are written to an Exodus II file. These coordinate frames are generally used as reference coordinate systems during analysis. In CUBIT, the user may define multiple exodus coordinate frames. When created, a coordinate frame is assigned an id. Exodus coordinate frames can be created using x-y-z coordinates, nodes or vertices with the following commands:

Exodus Create Coordinate Frame
<xval> <yval> <zval> //origin
<xval> <yval> <zval> //z-axis
<xval> <yval> <zval> //xz-plane
[tag { 'R' | 'C' | 'S' }]

Exodus Create Coordinate Frame Node
<node_origin_id>
<node_zaxis_id>
<node_xzplane_id>
[tag { 'R' | 'C' | 'S' }]

Exodus Create Coordinate Frame Vertex
<vertex_origin_id>
<vertex_zaxis_id>
<vertex_xzplane_id>
[tag { 'R' | 'C' | 'S' }]

Using the 'tag' option specifies the type of coordinate frame, i.e., rectangular (R), cylindrical (C) or spherical (S). The default coordinate frame type is rectangular. Exodus coordinate frames may also be listed and deleted using the commands below:

List Exodus Coordinate Frame [ids] [<frame_id>]

Delete Exodus Coordinate Frame [ids] [<frame_id>| all]

Any exodus coordinate frames that exist at the time the exodus file is exported will be written out in the exodus file.

Exodus II File Specification

Exodus II Manual

The full Exodus II manual is available from the web.

Element Block Definition Examples

Multiple Element Blocks

Multiple element blocks are often used when generating a finite element mesh. For example, if the finite element model consists of a block which has a thin shell encasing the volume mesh, the following block commands would be used:

Block 100 Volume 1
Block 100 Element Type Hex8
Block 200 Surface 1 To 6
Block 200 Element Type Shell4
Block 200 Attribute 0.01
Mesh Volume 1
Export Genesis `block.g'

This sequence of commands defines two element blocks (100 and 200). Element block 100 is composed of 8-node hexahedral elements and element block 200 is composed of 4-node shell elements on the surface of the block. The "thickness" of the shell elements is 0.01. The finite element code which reads the Genesis file (block.g) would refer to these blocks using the element block IDs 100 and 200. Note that the second line and the fourth line of the example are not required since both commands represent the default element type for the respective element blocks.

Surface Mesh Only

If a mesh containing only the surface of the block is desired, the first two lines of the example would be omitted and the Mesh Volume 1 line would be changed to, for example

Mesh Surface 1 To 6.

Two-dimensional Mesh

CUBIT also provides the capability of writing two-dimensional Genesis databases similar to FASTQ. The user must first assign the appropriate surfaces in the model to an element block. Then a Quad* type element may be specified for the element block. For example

Block 1 Surface 1 To 4
Block 1 Element Type Quad4

In this case, it is important for users to note that a two-dimensional Genesis database will result. In writing a two-dimensional Genesis database, CUBIT ignores all z-coordinate data. Therefore, the user must ensure that the Element Block is assigned to a planar surface lying in a plane parallel to the x-y plane. Currently, the Quad* element types are the only supported two-dimensional elements. Two-dimensional shell elements will be added in the near future if required.

Step-By-Step Tutorials

The purpose of this chapter is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a brief tutorial on the use of the software package. This chapter is designed to demonstrate step-by-step instructions for generating a simple mesh on a perforated brick.

The following activity demonstrates the basics of using CUBIT to generate and mesh a geometry. By following these steps, you will become familiar with the basics of the command-line and GUI interfaces without stopping for detailed explanations. All the commands introduced in this tutorial are documented in subsequent chapters on this manual.

Here are a few tips for the examples in the tutorial:

- Focus on the instructions preceded with "Step" numbers. These walk you through a series of explicit activities that describe how to complete the task.
- Refer to the screen shots and other pictures that show what you should see on your own monitor as you progress through the tutorial.
- In this tutorial, command line options will look like this:

  cubit> <Your commands go here>

If you do not have the Graphical User Interface (GUI) version of CUBIT, follow the steps in the right column below, otherwise, proceed through the steps on the left:

GUI	CL	
Overview	Overview	
Step 1	Step 1	Beginning Execution
Step 2	Step 2	Creating the Brick
Step 3	Step 3	Creating the Cylinder
Step 4	Step 4	Adjusting the Graphics Display
Step 5	Step 5	Forming the Hole
Step 6	Step 6	Setting Interval Sizes
Step 7	Step 7	Surface Meshing
Step 8	Step 8	Volume Meshing
Step 9	Step 9	Inspecting the Model
Step 10	Step 10	Defining Boundary Conditions
Step 11	Step 11	Exporting the Mesh

Additional Tutorials

Power Tools GUI Tutorial - A tutorial on geometry decomposition and cleanup using the Power Tools on the new CUBIT GUI.

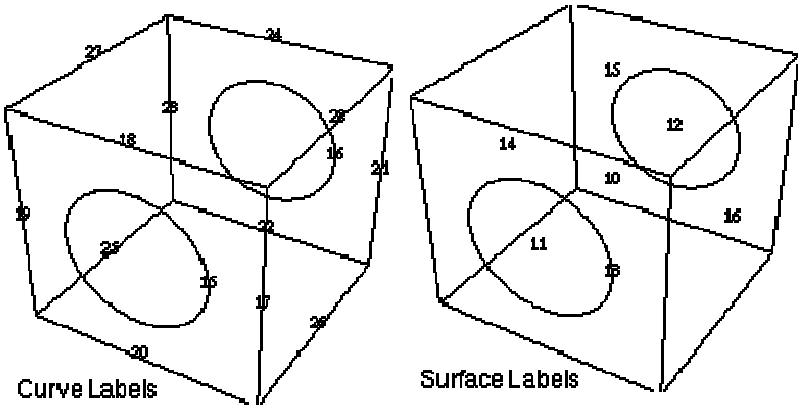
Command Line Basic Tutorial

Overview

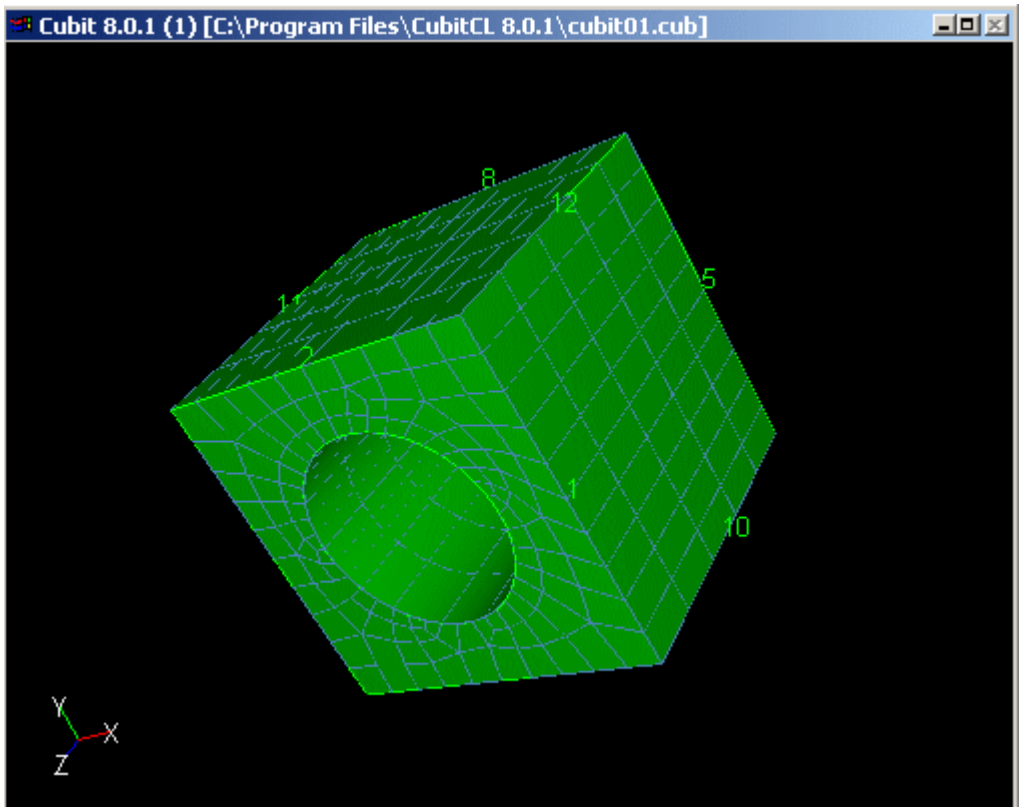
This tutorial demonstrates the use of CUBIT to create and mesh a brick with a through-hole. The primary steps in performing this task are:

- Creating the geometry
- Setting the interval sizes and meshing schemes
- Meshing the geometry
- Specifying the boundary conditions
- Exporting the mesh

Each of these steps is described in detail in the following sections. The geometry in this tutorial is a brick with a cylindrical hole in the center, shown in the figure below. This figure also shows the curve and surface identification (ID) numbers, which are referenced in the command lines shown with each step. The final meshed body is shown in the next figure.



Geometry for Cube with Cylindrical Hole

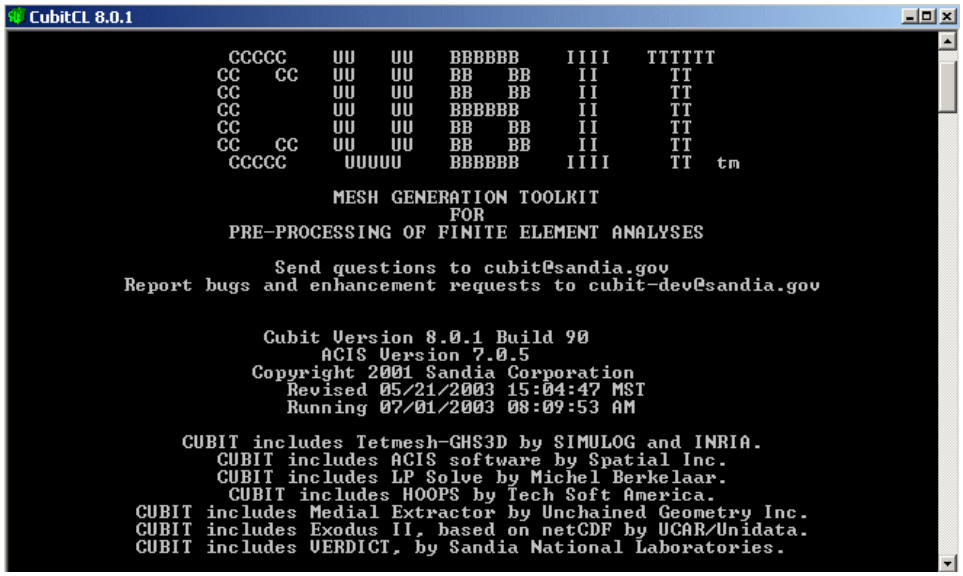


Generated Mesh for Cube with Cylindrical Hole

Command Line Basic Tutorial

Step 1: Beginning Execution

Type "cubit" from a UNIX prompt to begin execution of CUBIT. If you have not yet installed CUBIT, see instructions for doing so in the "CUBIT Installation" Appendix. A CUBIT console window will appear which tells the user which CUBIT version is being run and the most recent revision date. An example of the UNIX output window is shown below. This window echoes the commands and relays information about the success or failure of attempted actions.



Some things to notice are:

- At the top of the CUBIT window you will be told where the commands entered in this CUBIT session will be journaled. For example: "Commands will be journaled to `cubit03.jou' for this example.
- In addition to the CUBIT version, the code also reports the versions of ACIS and VTK that have been compiled into CUBIT.
- The command line prompt appears after the banner screen, and appears as "**CUBIT>**".
- Commands are entered at that prompt, followed by the "Enter" key.
- Upon startup, a graphics window should also appear, with an axis triad in the lower left hand corner (this window will not appear if CUBIT is started with the -nographics option.)

Command Line Basic Tutorial

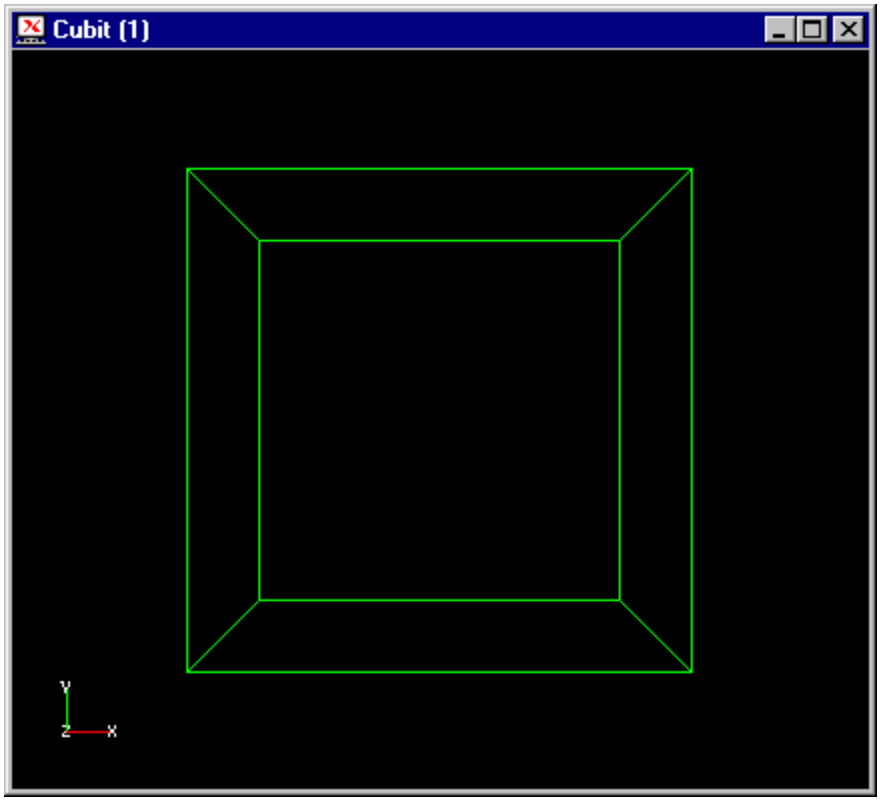
Step 2: Beginning Execution

Now you may begin generating the geometry to be meshed. You will create a brick of width 10, depth 10 and height 10. The width and depth correspond to the x and y dimensions of the object being created. The "width" or x-dimension is screen-horizontal and the "depth" or y-dimension is screen-vertical. The height or z-dimension is out of the screen. The command to create this object is:

cubit> create brick width 10 depth 10 height 10 (OR)

cubit> create brick x 10

The cube should appear in your display window as shown below:



Display of Brick

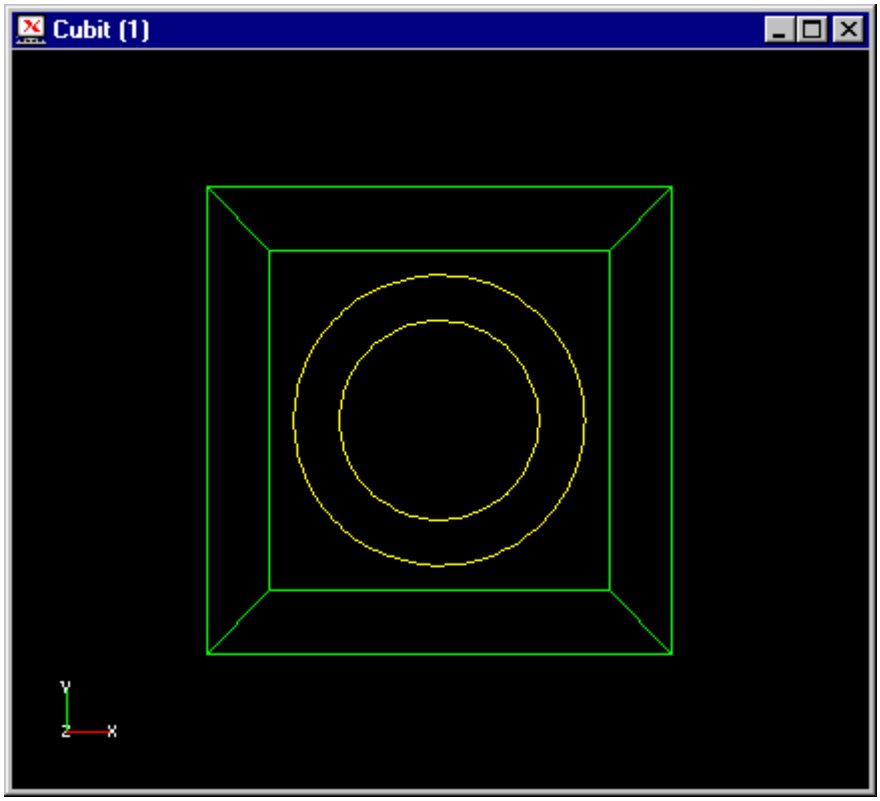
Command Line Basic Tutorial

Step 3: Creating the Cylinder

Now you must form the cylinder which will be used to cut the hole from the brick. This is accomplished with the command:

```
cubit> create cylinder height 12 radius 3
```

At this point you will see both a brick and a cylinder appear in the CUBIT display window, as shown below:



Brick and Cylinder

Command Line Basic Tutorial

Step 4: Adjusting the Graphics Display

The geometry is drawn in the graphics display in perspective mode by default from a viewing direction of the +z axis. This view can now be adjusted to verify the proper orientation of the geometry just created. The orientation of the geometry can be adjusted using the command line or interactively with the mouse.

Command Line

You can adjust the orientation of the object from the command line. For example, the from command can be used as follows

```
cubit>from 20 15 25

cubit>display
```

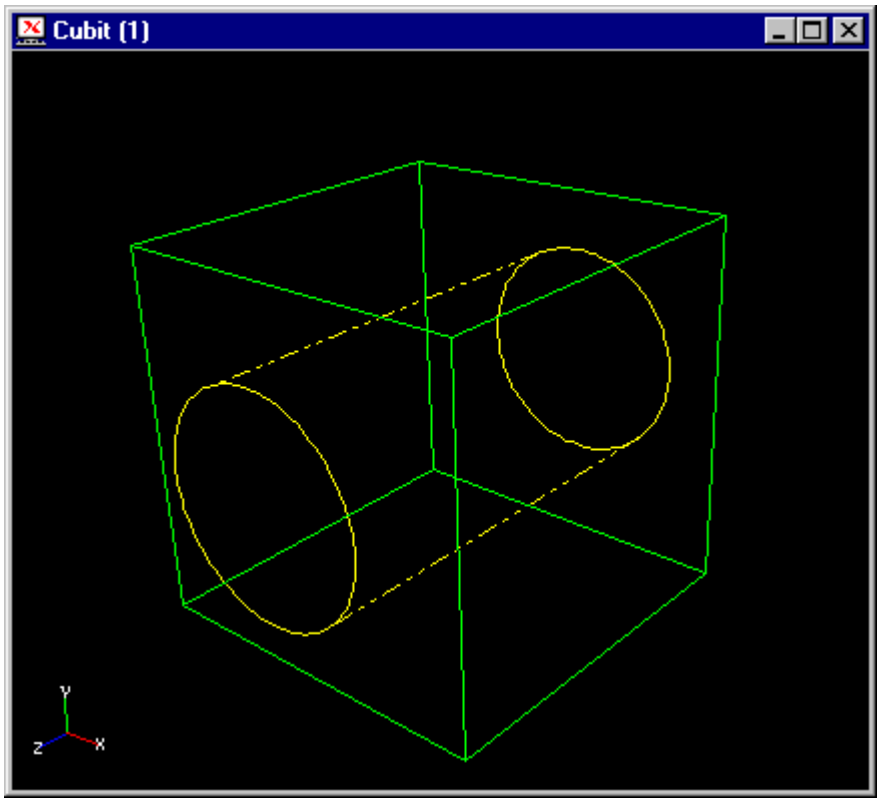
Mouse

To interactively change the orientation, activate your graphics window by placing your cursor in the window or by clicking at the top of it (this will vary depending upon your window settings in your operating system).

- Use the **left mouse button** to interactively rotate the view
- Use the **middle mouse button** to zoom in or out

- Use the **right mouse button** to pan the view.

Use the mouse buttons to make the display look the figure below:



View from a Different Perspective

Command Line Basic Tutorial

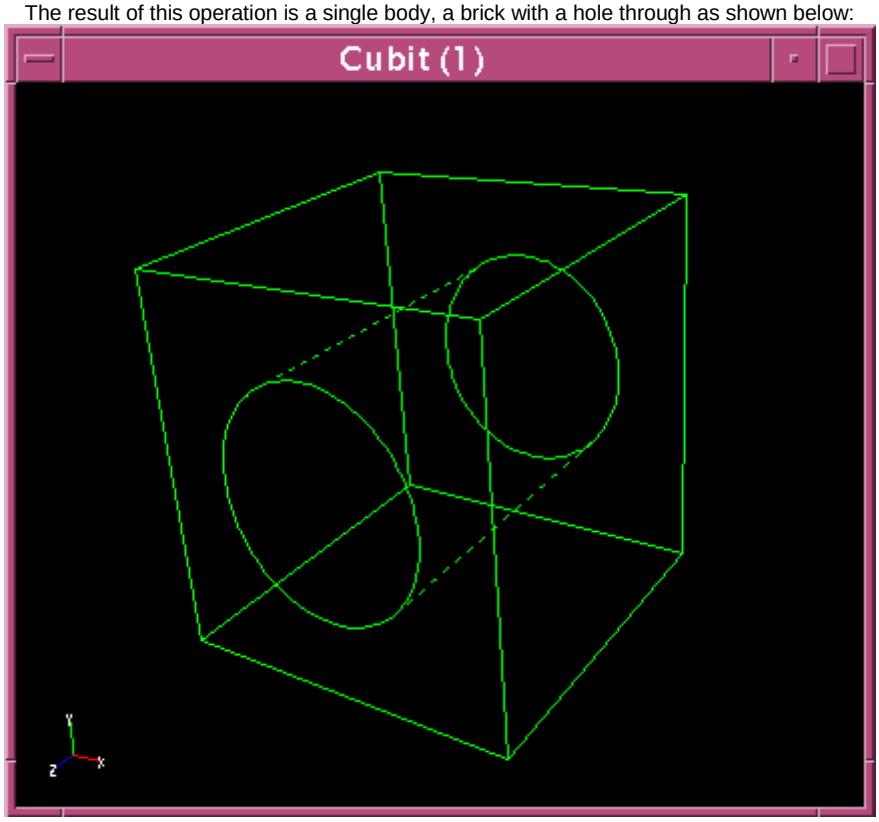
Step 5: Forming the Hole

Now, the cylinder can be subtracted from the brick to form the hole in the block. Issue the following command:

cubit> subtract 2 from 1

Note that both original bodies are deleted in the Boolean operation and replaced with a new body (with an id of 1) which is the result of the Boolean operation Subtract .

The result of this operation is a single body, a brick with a hole through as shown below:



Brick after Subtracting the Cylinder

We have now completed creating the geometry, and are ready to generate a mesh.

Command Line Basic Tutorial

Step 6: Setting Interval Sizes

The volume shown in Step 5 will be meshed by sweeping a surface mesh from one side of the brick to the other. Before generating any mesh, the user must specify the size of the elements to be generated. In this example, one element size will be specified for the volume as a whole and a smaller size will be specified for around the hole. A direct interval setting will be specified for the sweep direction.

To set the interval size for the overall body, enter the command

cubit> body 1 size 1.0

Since the brick is 10 units in length on a side, this specifies that each straight curve is to receive approximately 10 mesh elements.

In order to better resolve the hole in the middle of the top surface, we set a smaller size for the curve bounding this hole. To find the id number of the curve bounding the hole, the user can either pick the curve (See Selecting Entities with the Mouse) or turn curve labels on and regenerate the view. To do the latter, use the command

cubit> label curve on

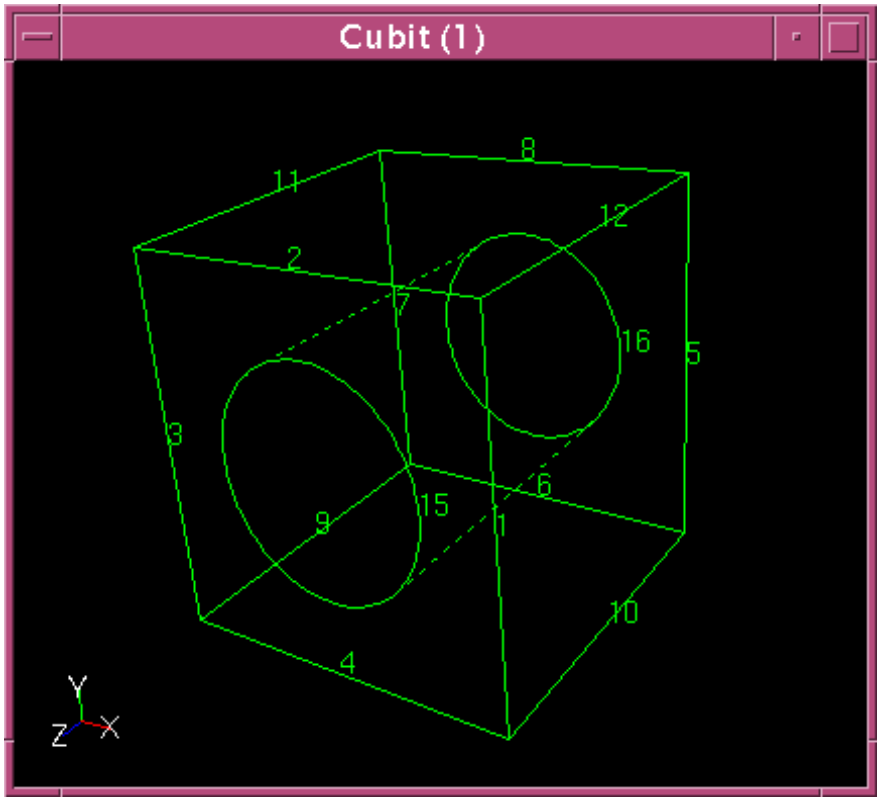
cubit> display

The default size of the labels can sometimes be too small to read. To change the text size, use the graphics text size command:

cubit> graphics text size 2

cubit> display

The result is shown in the figure below. Then the interval size can be set for the appropriate curve:



Geometry with Curve Labeling Turned on

cubit> curve 15 interval size 0.5

Finally, we would like to generate exactly 5 element layers in the sweep direction. This is accomplished by setting the intervals on curve 11:

cubit> curve 11 interval 5

Command Line Basic Tutorial

Step 7: Surface Meshing

Now that all the necessary intervals have been set, the meshing can proceed. Begin by meshing the front surface (with the hole) using the paving algorithm. This is done in two steps. First, set the scheme for that surface to Pave; then, issue the command to Mesh. Since the surface to be paved is number 11, issue the command:

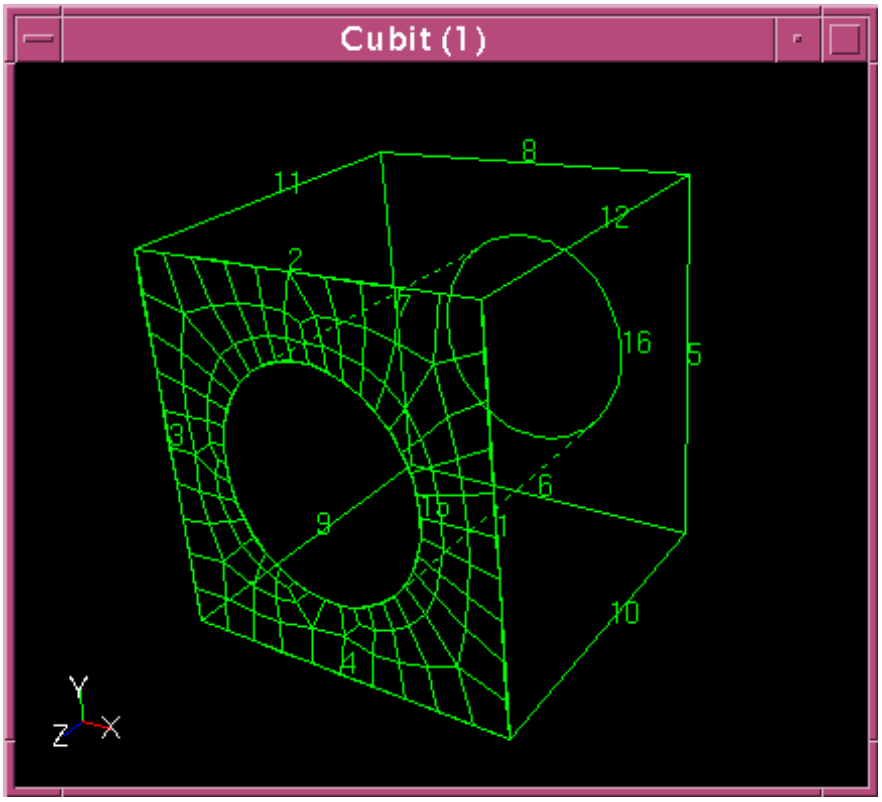
cubit> surface 11 scheme pave

With the meshing scheme specified, we proceed to mesh the surface:

cubit> mesh surface 11

cubit>display

A wire frame view of the result is shown below:



Surface Meshed with Paving

Command Line Basic Tutorial

Step 8: Surface Meshing

The volume mesh can now be generated. Again, the first step is to specify the type of meshing scheme to be used and the second step is to issue the order to mesh. In certain cases, the scheme can be determined by CUBIT automatically. For sweepable volumes, the automatic scheme detection algorithm also identifies the source and target surfaces of the sweep automatically.

To instruct the code to automatically determine the meshing scheme and in this case the source and target surfaces, enter the command:

cubit> volume 1 scheme auto

To view the results of auto scheme selection, certain data about the volume can be listed:

cubit> list volume 1

The results of this command are shown below; note that the scheme, and in this case the source and target surfaces, are reported toward the top of the list output.

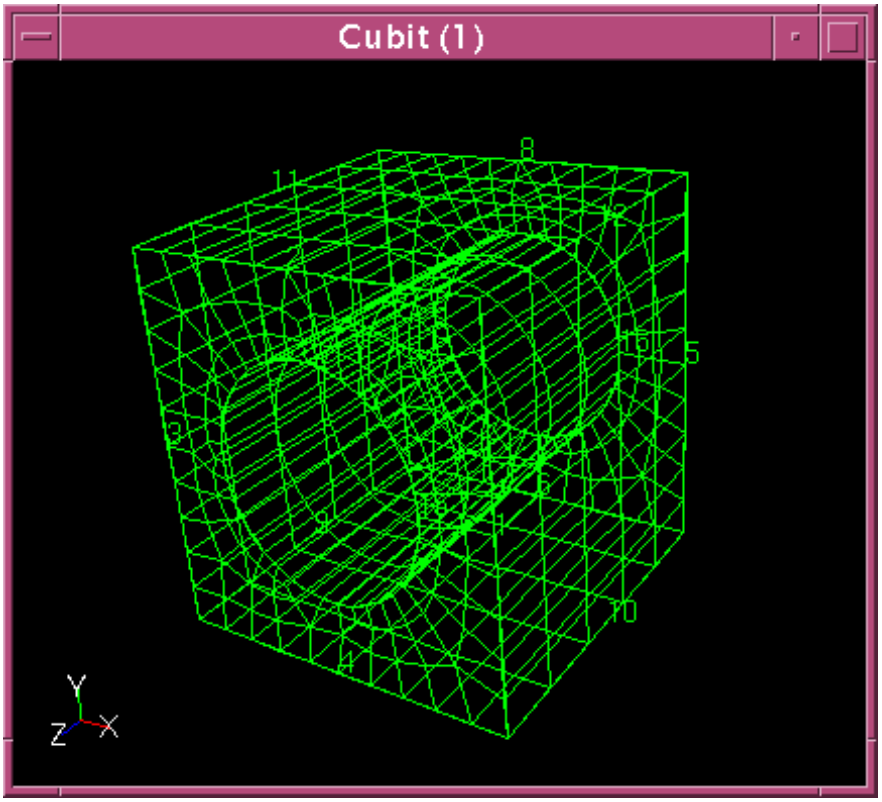
```
CubitCL 8.0.1

CUBIT> list volume 1
Volume Entity  (Id = 1)
        Visible:         Yes
        Meshed:          No
        Mesh Scheme:     sweep   (automatically selected)
          Source:        Surface 11,   (Ids = 11)
          Target:        Surface 12,   (Ids = 12)
          Sweep Smooth Scheme: Auto
        Smooth Scheme:   equipotential fixed

        Interval Count: 1
        Interval Size:  1.000000
        Block Id:       0

        7 Owned Surfaces:        Mesh Scheme                      Interval:
        ______Name_______     Id    +is meshed      Smooth Scheme Count    Size
              Surface 10     10          map-       winslow fixed     1       1
                   Periodic Interval: 10, Soft
              Surface 11     11         pave+       winslow fixed     1       1
              Surface 12     12         pave-       winslow fixed     1       1
               Surface 3      3          map-       winslow fixed     1       1
               Surface 4      4          map-       winslow fixed     1       1
               Surface 5      5          map-       winslow fixed     1       1
               Surface 6      6          map-       winslow fixed     1       1

        In Body 1.
Journaled Command: list volume 1

CUBIT> _
```

Output from Listing Volume 1

With the scheme set, the mesh command may be given:

cubit> mesh volume 1

The final meshed body will appear in the display window, as shown below:



Wireframe View of Volume Mesh

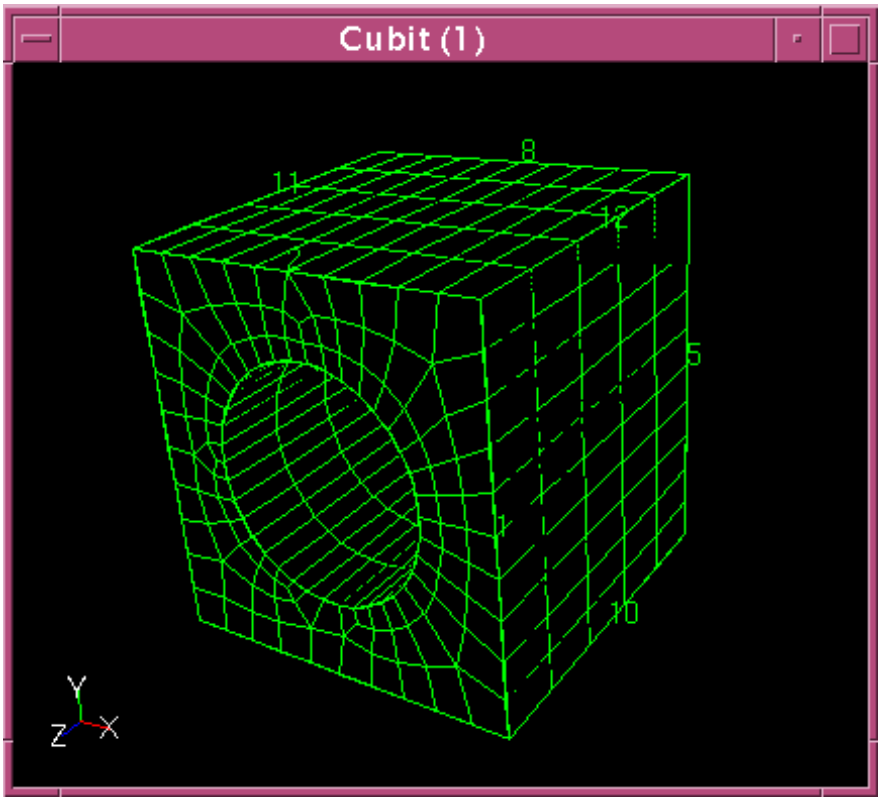
Command Line Basic Tutorial

Step 9: Inspecting the Model

The type, quality, and speed of rendering the image can be controlled in CUBIT by using several graphics mode commands, such as Wire Frame, Hidden Line, and Smooth Shade . For example:

cubit> graphics mode hiddenline

The hidden line display is illustrated below:

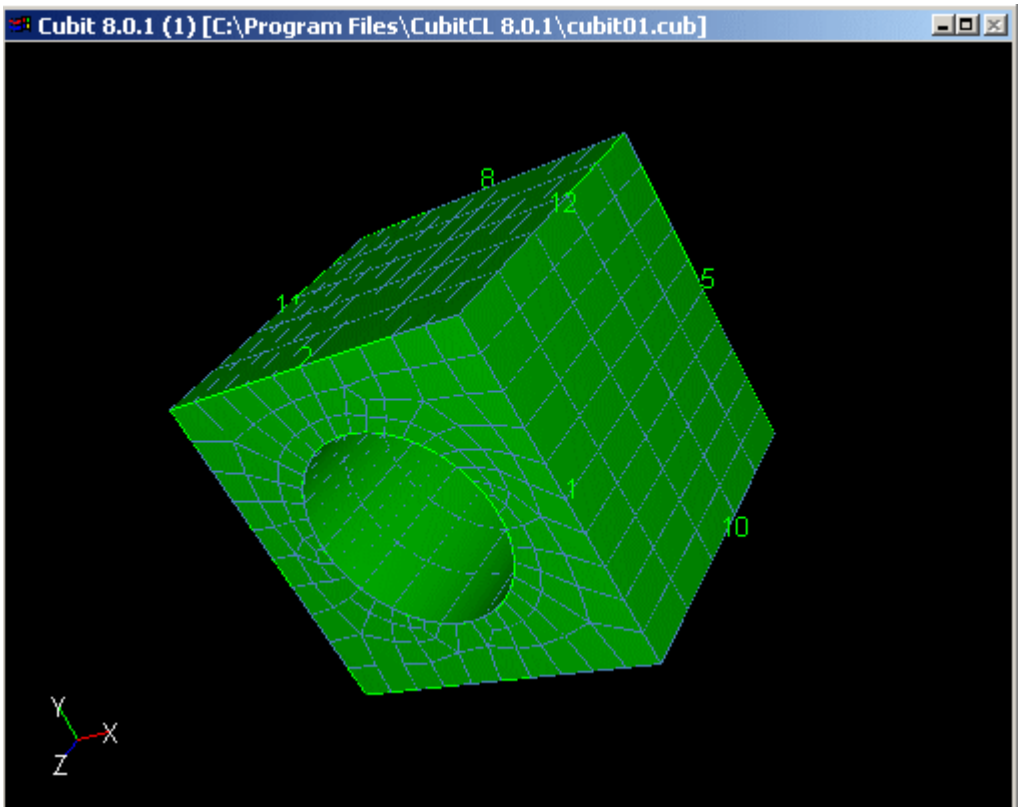


Hidden Line View of Mesh

Next, try:

cubit> graphics mode smoothshade

The smooth shade display is shown below. For detailed information on the viewing mode options, See Graphics Modes.



Smooth Shade View of Mesh

Although CUBIT automatically computes limited quality metrics after generating a mesh and warns the user about certain cases of bad quality, it is still a good idea to inspect a broader set of quality measures. To do this, enter the command:

```
cubit> quality volume 1
```

The results of the quality output are shown below. For an explanation of quality metrics along with acceptable ranges, see Mesh Quality Assessment. For the purposes of this tutorial, you can assume the quality metrics shown below are in an acceptable range.

```
CubitCL 8.0.1
CUBIT> quality volume 1

Volume 1 Hex quality, 580 elements:
-----------------------------------
   Function Name    Average      Std Dev      Minimum   (id)    Maximum  (id)
 ----------------  ---------    ---------    -------------    -------------
           Shape   5.465e-001   1.639e-001   2.036e-001 (18)   8.237e-001 (180)
-----------------------------------

Journaled Command: quality volume 1

CUBIT> _
```

Quality Table from Volume 1's Hex Mesh

Command Line Basic Tutorial

Step 10: Defining Boundary Conditions

Let us assume that we need to define one material type for the entire mesh, and a single node-based boundary condition on all surfaces. This is accomplished by identifying an Element Block and a Nodeset, respectively; the id numbers assigned to these entities are assigned by the user, usually by some convention meaningful to the analysis to be done. The element block and nodeset are identified using the commands:

```
cubit> block 100 volume 1

cubit> nodeset 100 surface all in volume 1
```

Command Line Basic Tutorial

Step 11: Exporting the Mesh

Finally, the mesh needs to be written to an ExodusII file. This is easily done:

```
cubit> export genesis `brick_with_hole.g'
```

The filename and extension are arbitrary and, like the block and nodeset numbers, are usually named according to a convention meaningful to the analysis.

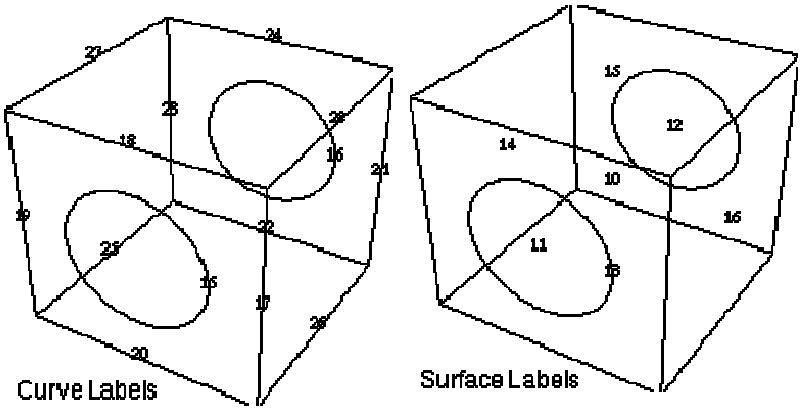
GUI Basic Tutorial

Overview

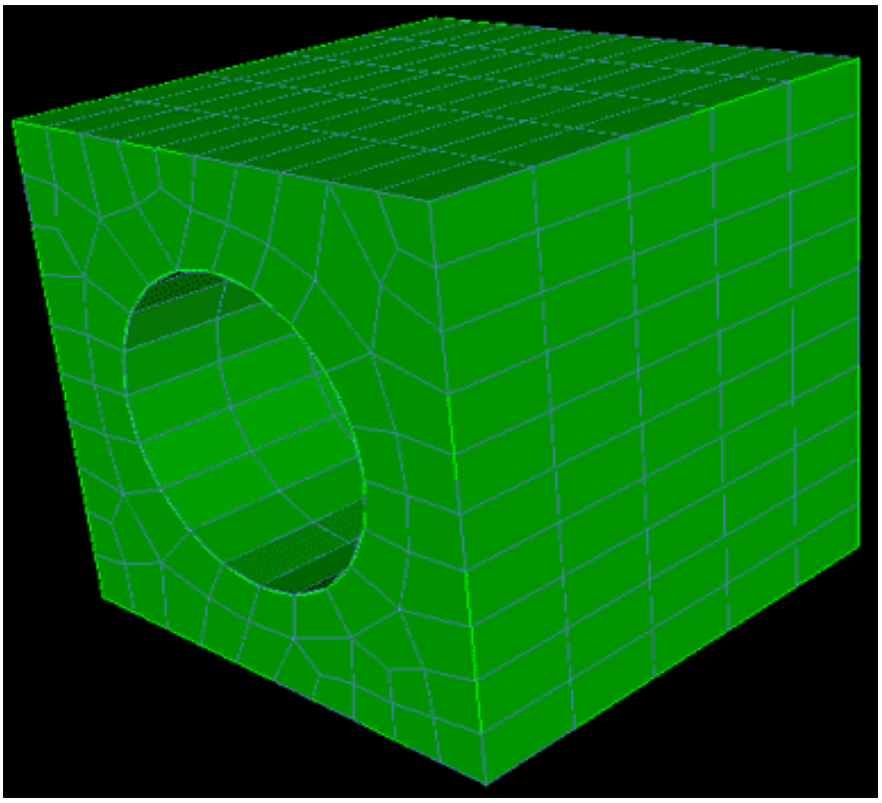
This tutorial demonstrates the use of CUBIT to create and mesh a brick with a through-hole. The primary steps in performing this task are:

- Creating the geometry
- Setting the interval sizes and meshing schemes
- Meshing the geometry
- Specifying the boundary conditions
- Exporting the mesh

The geometry for this tutorial is a brick with a cylindrical hole in the center, shown in the figure below. This figure also shows the curve and surface identification (ID) numbers, which are referenced in the command lines options shown with each step. The final meshed body is shown in the next figure.



Geometry for Brick with Cylindrical Hole

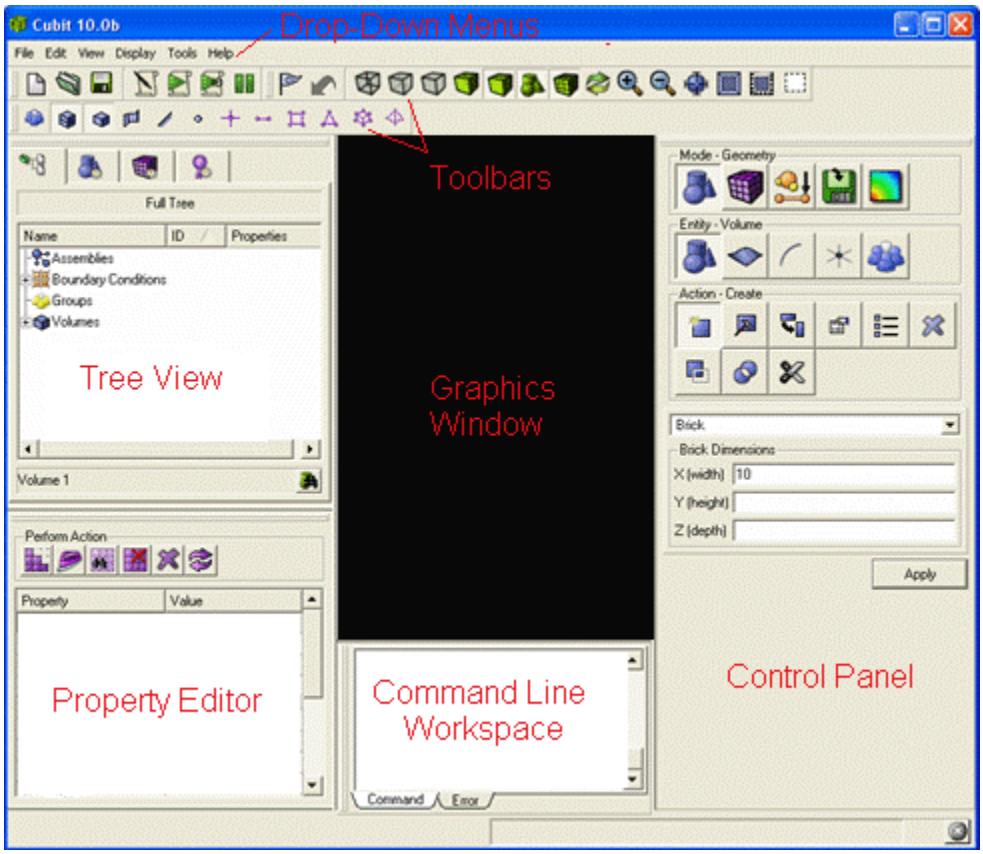


Generated Mesh for Brick with Cylindrical Hole

GUI Basic Tutorial

Step 1: Beginning Execution

Type "**cubit**" from a UNIX prompt or select cubit from the start menu if you are running on a PC with Windows. If you have not yet installed CUBIT, see instructions for doing so in the "CUBIT Installation" Appendix. The CUBIT Application Window will appear as illustrated below:



CUBIT Application Window

The use of each windows in the CUBIT program is described briefly below

Graphics Window	The current model will be displayed here. Zooming, panning, and rotating are also performed in this window.
Pull-down Menus	Functions such as file management, edit controls, display options, user preferences, journal file management, window manipulation, and help are available in the pull-down menus.
Tool Bars	A large selection of selectable icons duplicate the functions found in the pull-down menus. Additionally, tools to change pick type, hide/show the geometry tree, and temporarily change the behavior of the left mouse button are available.
Tree View	A graphical representation of the current solid model's relationships between geometric entities. Selecting an entity in this window will also select the entity in the graphics window. The Tree View also contains the geometry repair power tools, meshing power tools, and mesh quality power tools.
Command Line	The command line workspace contains both the cubit command and error windows.

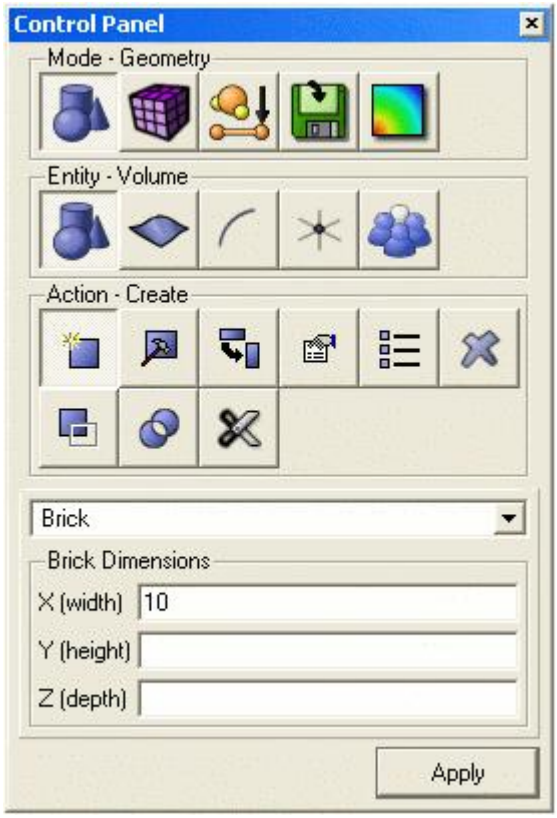
Workspace	The command window is used to enter cubit commands and view the output. The error window is used to view cubit errors..
Control Panel	Most Cubit commands are available in the control panel. The control panel is organized topologically, by mode.
Property Editor	A list of properties of the selected geometry, mesh, or boundary condition entity. Some of the properties can also be edited from this window.

GUI Basic Tutorial

Step 2: Creating the Brick

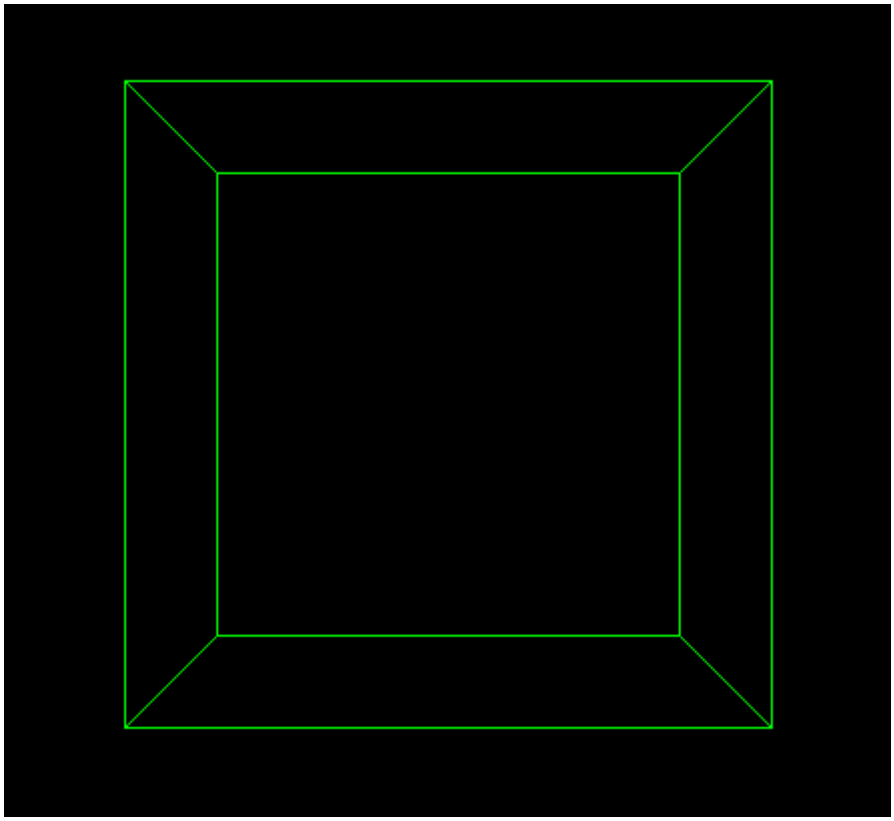
Now you may begin generating the geometry to be meshed. You will create a brick of width 10, depth 10 and height 10. The width and depth correspond to the x and y dimensions of the object being created. The "width" or x-dimension is screen-horizontal and the "depth" or y-dimension is screen-vertical. The height or z-dimension is out of the screen.

- On the Command Panel, select **Geometry**, then **Volume**, then **Create**. Brick is the default type.



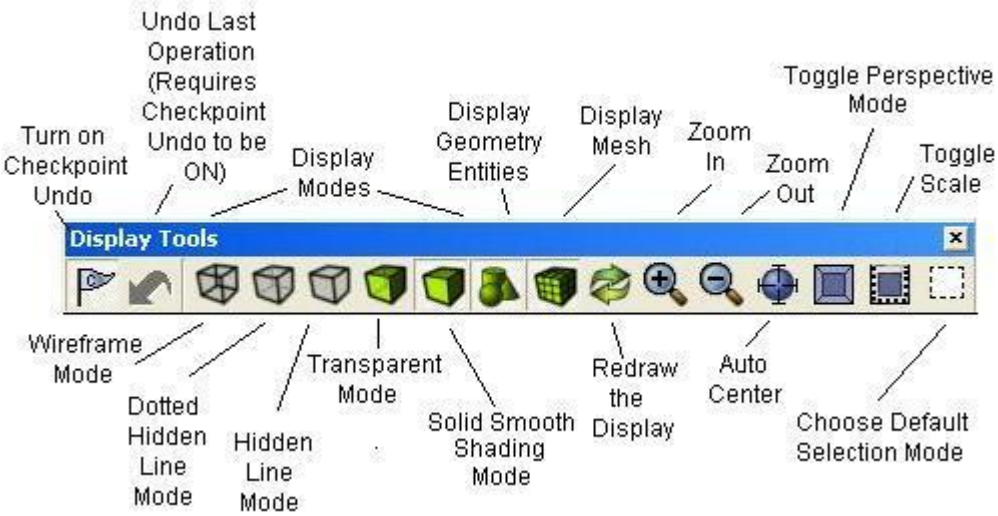
- Enter values for X, Y, and Z. Note, X (width) has a default value of 10. Select **Apply** to create the brick.

The brick should appear in your Graphics window as shown below.



Display of Brick

If you would like to change the rendering mode of your model, you may click on the wire frame button or one of the other view buttons in the Display Tools tool bar.

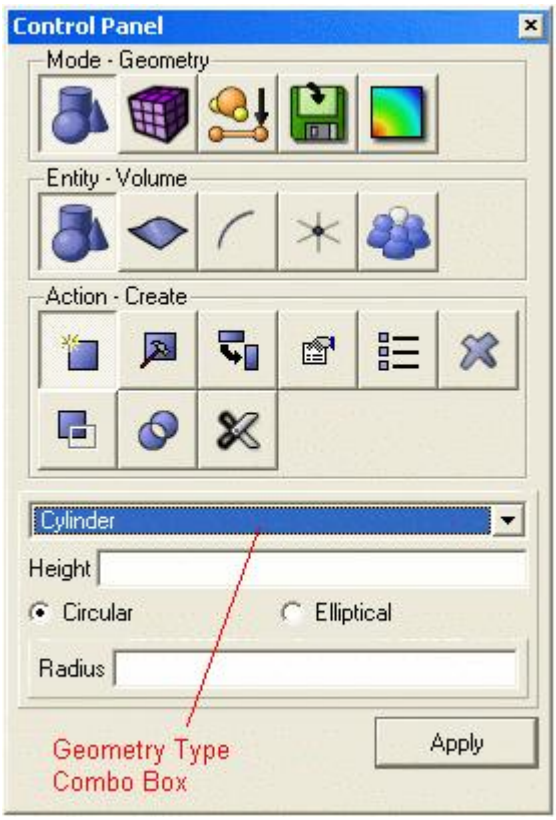


GUI Basic Tutorial

Step 3: Creating the Cylinder

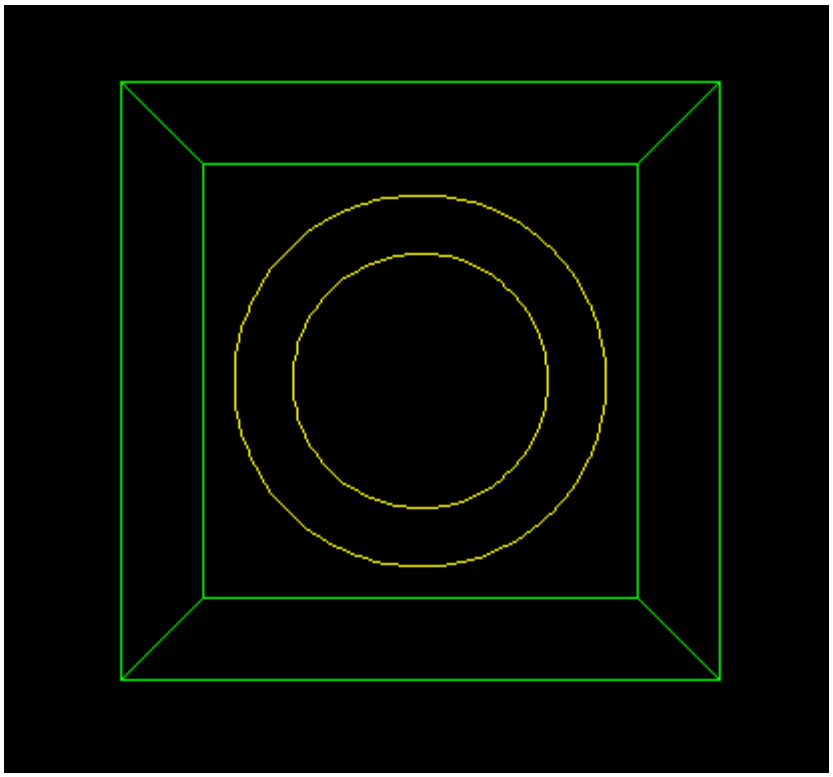
Now you must form the cylinder which will be used to cut a hole in the brick.

- Select **Cylinder** from the **Create** combo box.



- Enter **12** for the height and **3** for the radius. Then select **Apply**.

The brick and the cylinder should appear in your display window as shown below:



Brick and Cylinder

GUI Basic Tutorial

Step 4: Adjusting the Graphics Display

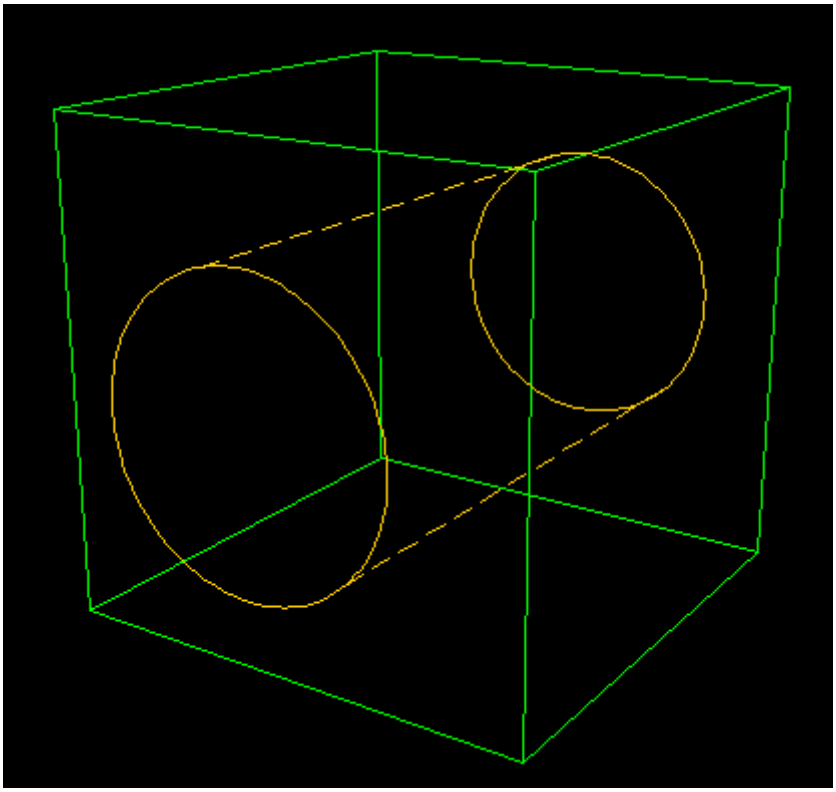
The geometry is drawn in the graphics window in perspective mode, by default from a viewing direction of the +z axis. This view can now be adjusted to verify the proper orientation of the geometry just created.

The following button clicks apply for 3-button mice (these are the default GUI settings):

- **left** will pick when the mouse is over an entity. Left click will also pan when held down.
- **middle** will rotate
- **right** will show a context menu when an entity is selected. Right click will zoom when no entity is selected.

Mouse button behavior can be customized from the Tools-Options menu for use with non 3-button mice.

Use the mouse buttons to make the display look like the figure below.



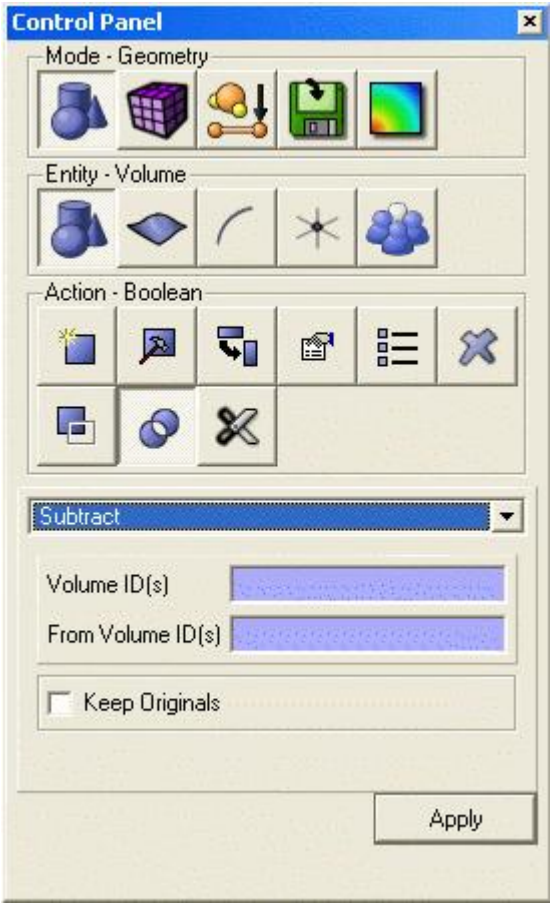
View from Different Perspective

GUI Basic Tutorial

Step 5: Forming the Hole

Now the cylinder can be subtracted from the brick to form the hole in the block.

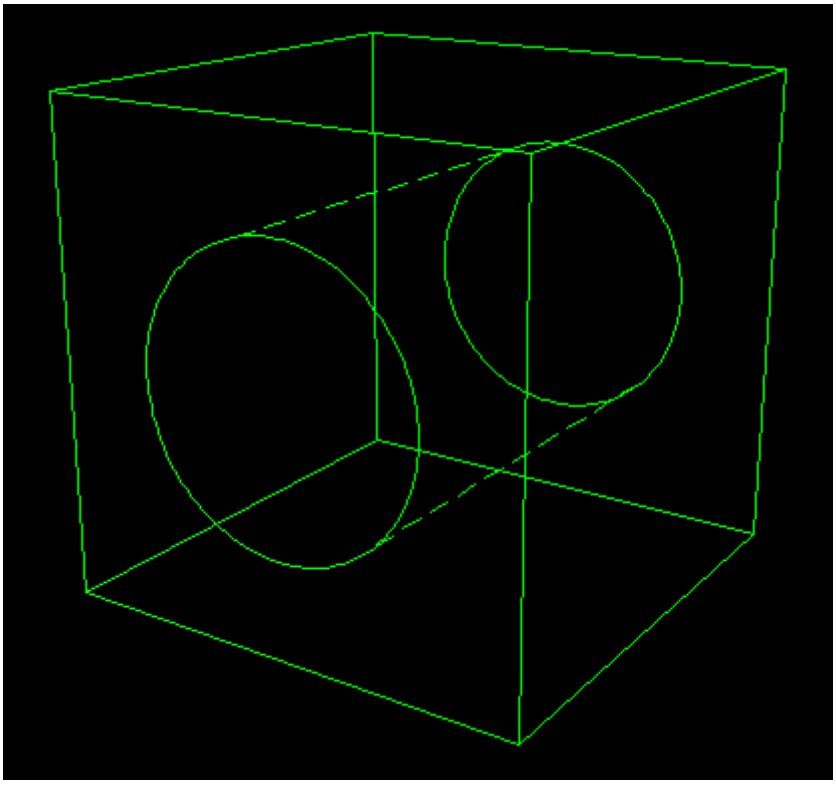
- Select the **Boolean** action button. Then select **Subtract** from the Boolean combo box.



- Enter **2** for **Subtract Volume ID(s)** and **1** for **From Volume ID(s)**.
- Select the **Apply** button

You can also select the brick or cylinder interactively. Place the cursor in the **Subtract Volume ID(s)** field and click. This field is known as a **Pick Widget**. Clicking in a pick widget automatically sets the graphics pick mode for the entity type expected by the pick widget. Move the cursor to the graphics window and, using the left mouse button, select an entity. The id of the selected entity will be echoed into the pick widget field. Holding the control key while selecting entities in the graphics window will select multiple entities.

Notice that both original volumes are deleted in the Boolean operation and replaced with a new volume, with an id of 1. The result of this operation is a single volume, a brick with a hole through it, as shown below.



Brick after Subtracting Cylinder

We have now completed creating the geometry, and are ready to generate a mesh.

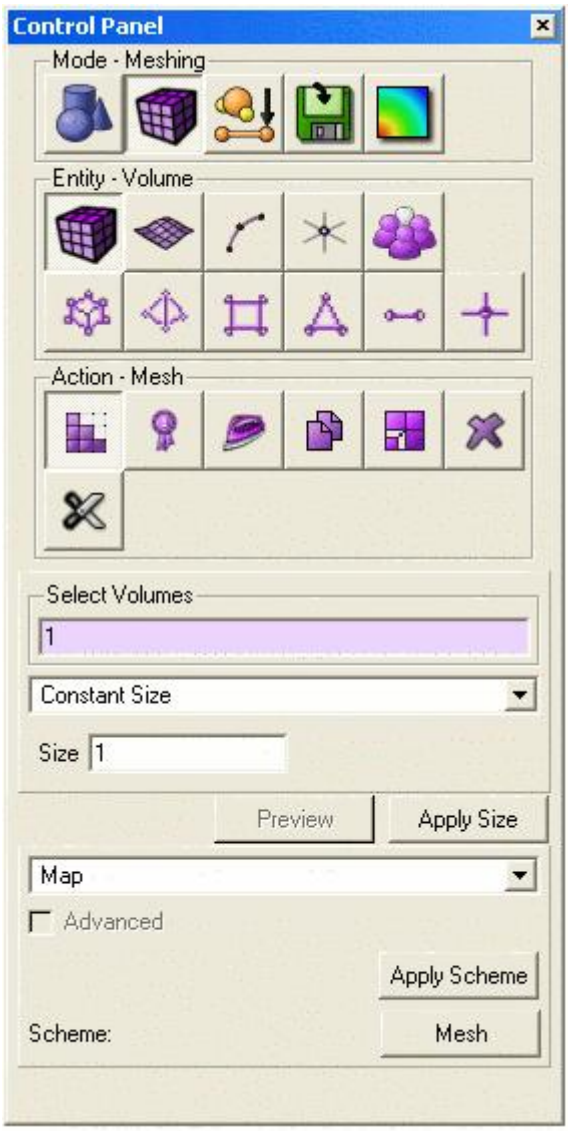
GUI Basic Tutorial

Step 6: Setting Interval Sizes

The volume shown in Step 5 will be meshed by sweeping a surface mesh from one side of the brick to the other. Before generating any mesh, the user must specify the size of the elements to be generated. In this example, one element size will be specified for the volume as a whole and a smaller size will be specified for around the hole. A direct interval setting will be specified for the sweep direction.

To set the interval size for the overall volume, do the following:

- Change the mode to **Meshing**, then select **Volume** followed by **Meshing**.

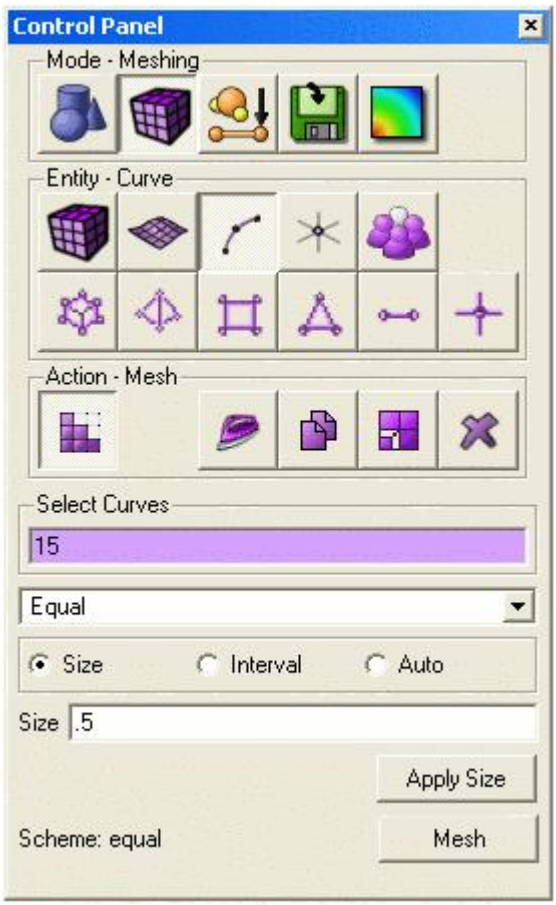


- Place the cursor into the **Select Volumes** field. Since this is a pick widget, click anywhere on the volume in the graphics window. Alternatively, type 1 in the field. Set the **Interval Size** to **1.0** and select **Apply Size**

Since the brick is 10 units in length on a side, this specifies that each straight curve is to receive approximately 10 mesh elements.

In order to better resolve the hole in the middle of the top surface, we set a smaller size for the curve bounding this hole.

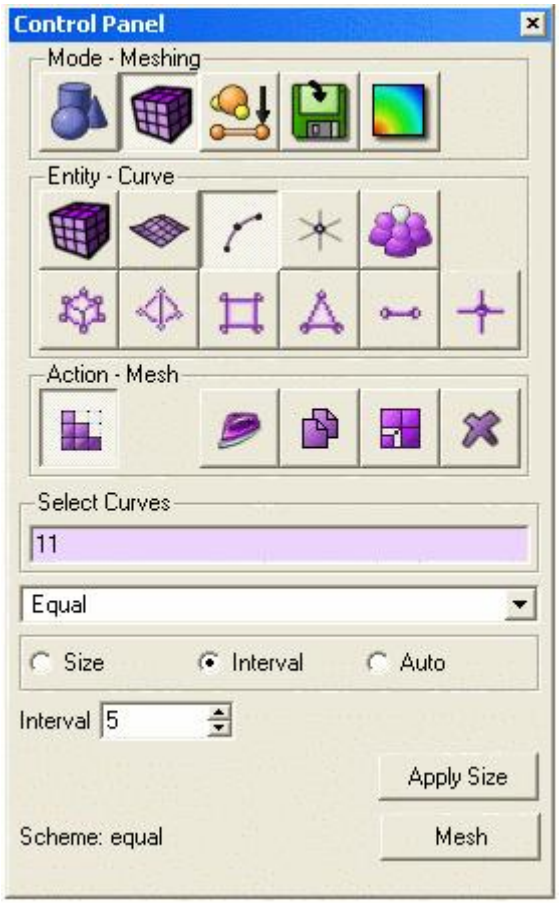
- Change the object of the command panel to curve by selecting **Curve** from the **Entity** buttons.



- Place the cursor into the **Select Curves** pick widget field. Select the near end of the cylinder in the graphics window. Once you have selected the curve, the id of that curve, **15** should appear in the Selected Curves field. Select the **Size** radio button
- Enter **0.5** for the size and select **Apply Size**.

Finally, we would like to generate exactly 5 element layers in the sweep direction. This is accomplished by setting the intervals on one of the curves in the sweep direction.

- Place the cursor back into the **Selected Curves** field and enter **11**.
- Select the **Interval** radio button
- Enter an interval count of **5** and select **Apply**.

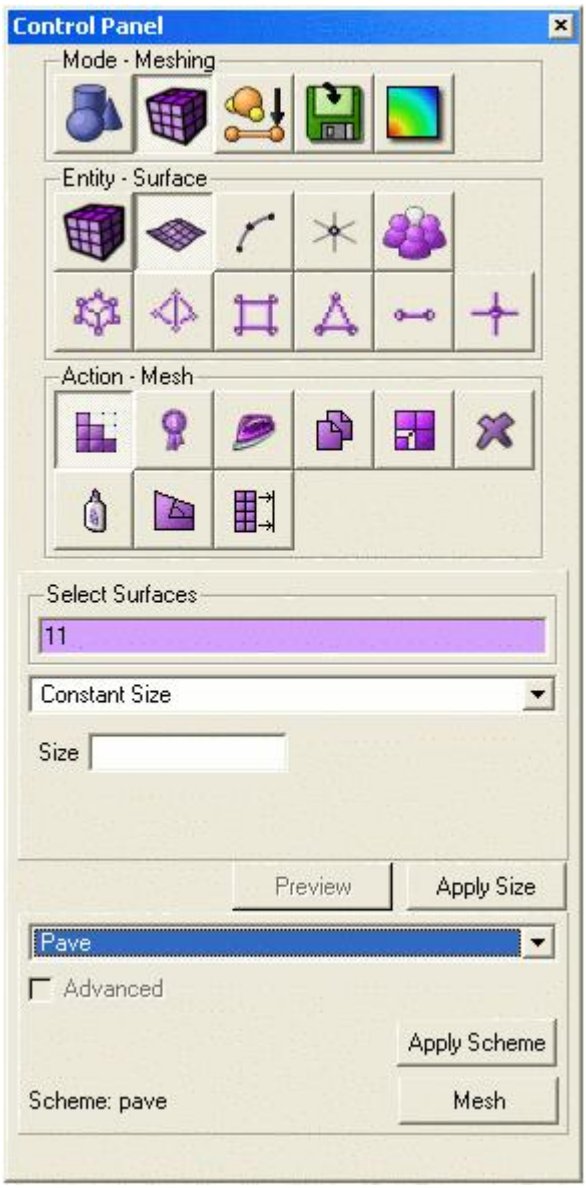


GUI Basic Tutorial

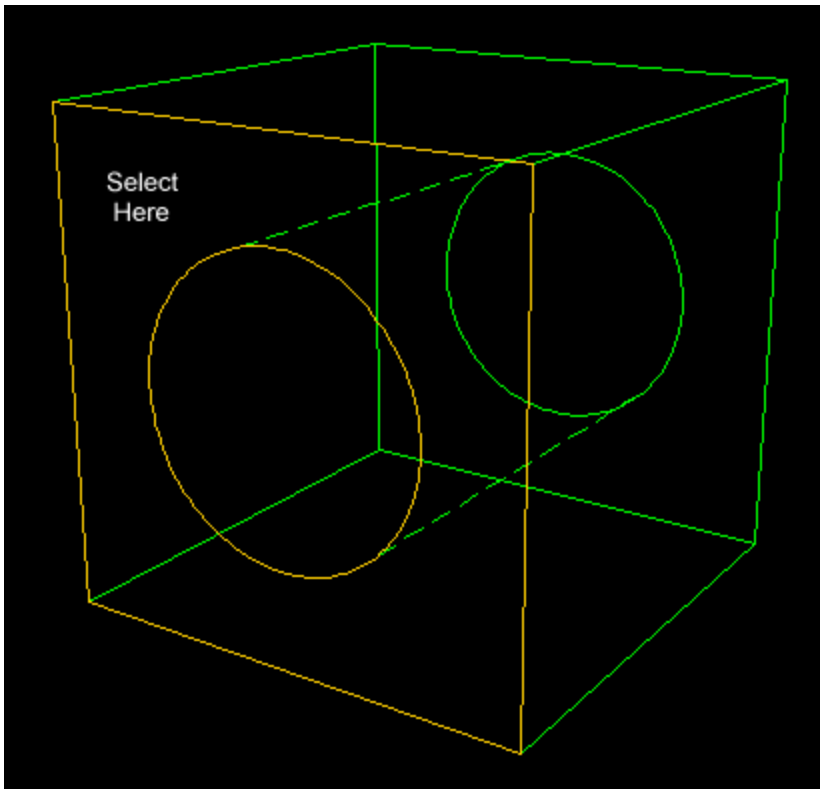
Step 7: Surface Meshing

Now all necessary intervals have been set, the meshing can proceed. Begin by meshing the front surface (with the hole) using the paving algorithm. This is done in two steps. First, set the scheme for that surface to Pave, then issue the command to Mesh.

- Select **Surface** then **Mesh** buttons in the Control Panel.
- Select **Pave** in the **Available Mesh Schemes** combo box.

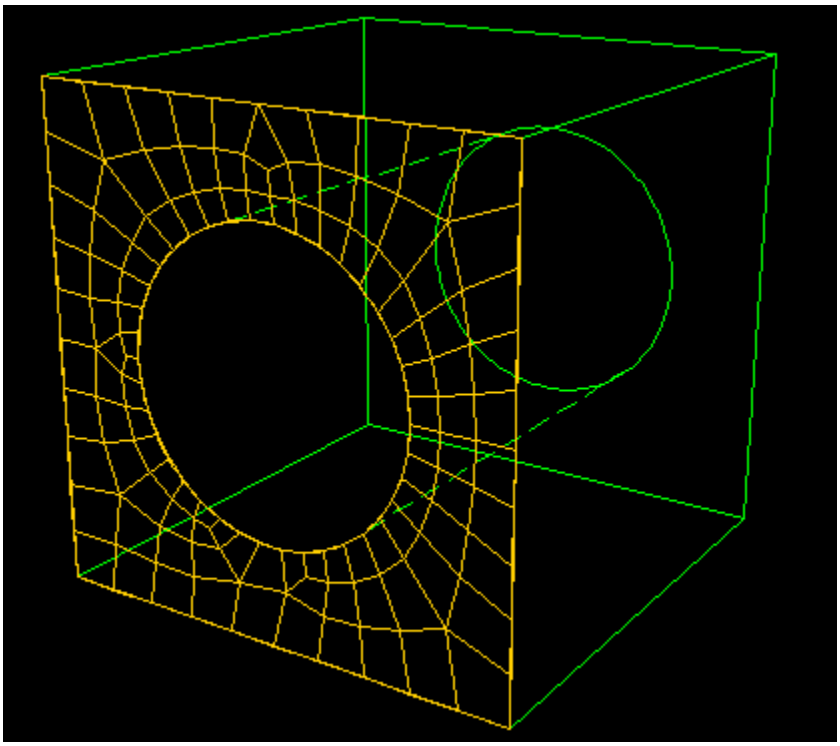


Place the cursor into the **Surface ID(s)** field. Select the front surface of the object by selecting anywhere within the region indicated. The id of **Surface 11** will be echoed in the field.



- Select the **Apply** button to set the scheme.
- Select the **Mesh** button to mesh the surface.

A mesh should be generated on surface 11 using the paving algorithm. The result is shown below.



Surface Meshed with Paving

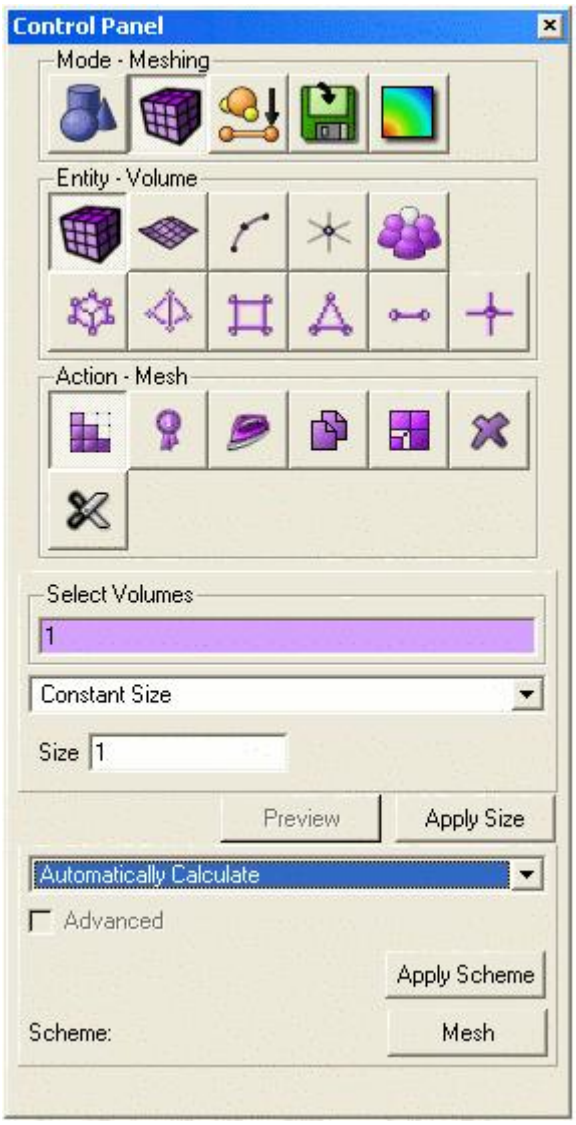
GUI Basic Tutorial

Step 8: Volume Meshing

The volume mesh can now be generated. Again, the first step is to specify the type of meshing scheme that should be used and the second step is to issue the order to mesh. In certain cases, the scheme can be determined by CUBIT automatically. For sweepable volumes, the automatic scheme detection algorithm also identifies the source and target surfaces of the sweep automatically.

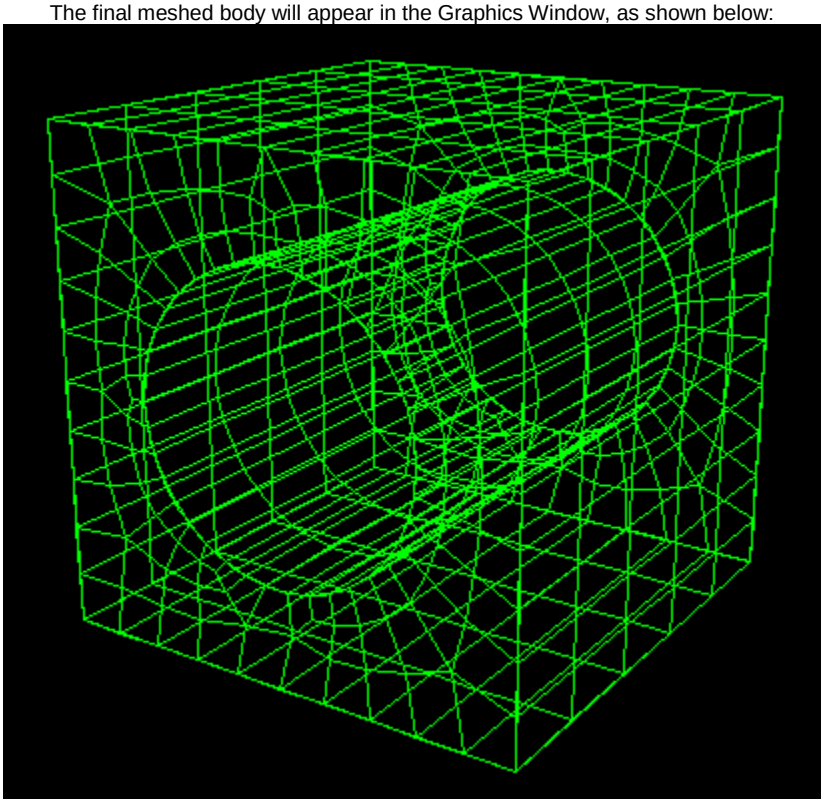
To instruct the code to automatically determine the meshing scheme, and in this case the source and target surfaces, do the following:

- Select **Volume** then **Mesh** on the control panel.



- Place the cursor into the **Volume ID(s)** field then select the volume in the Graphics Window. The id of **Volume 1** should appear in the field. Choose the **Automatically Calculate** scheme using the combo box provided.
- Select **Apply Scheme** to set the scheme. Then select **Mesh** to mesh the volume.

The final meshed body will appear in the Graphics Window, as shown below:

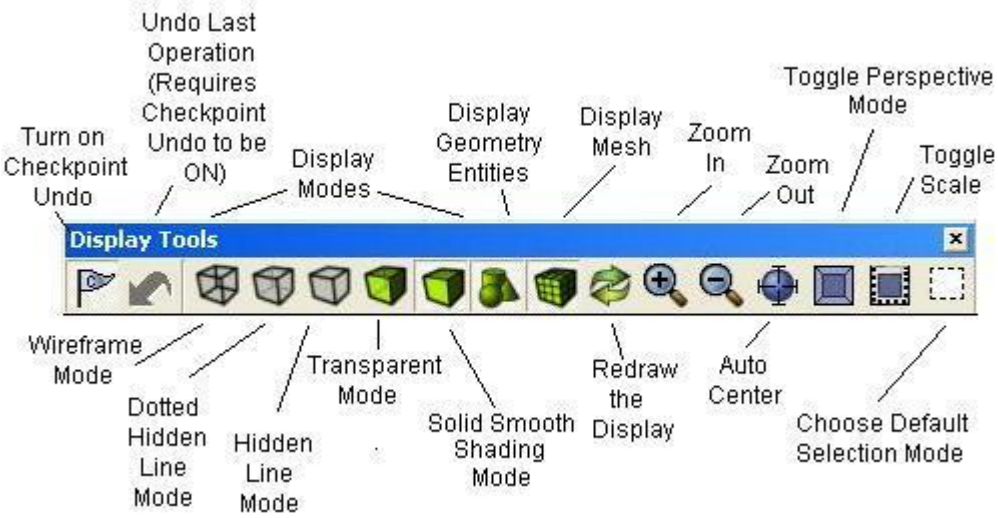


Wire Frame View of Volume Mesh

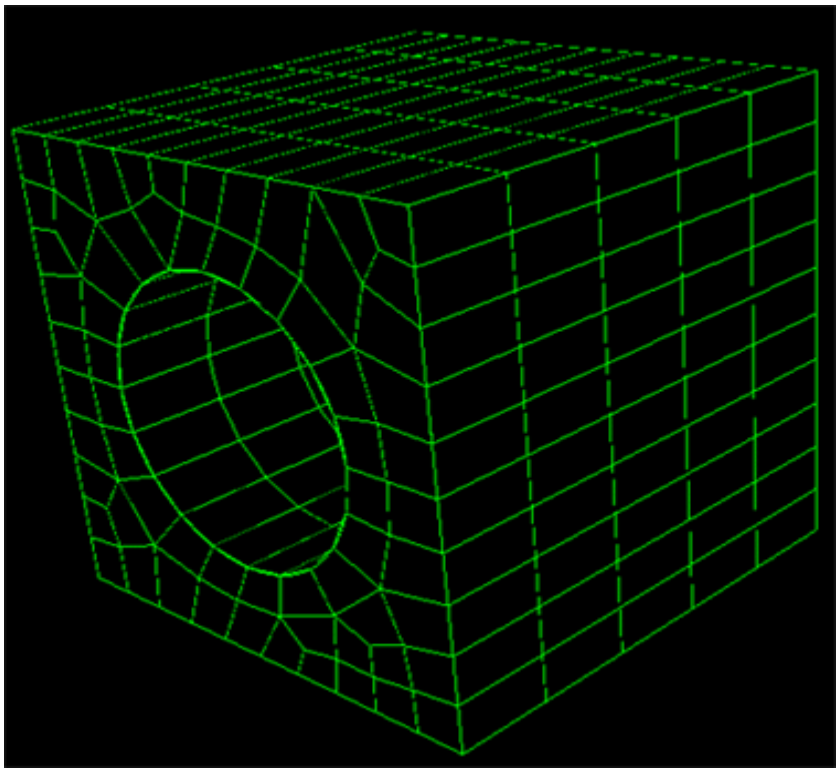
GUI Basic Tutorial

Step 9: Inspecting the Model

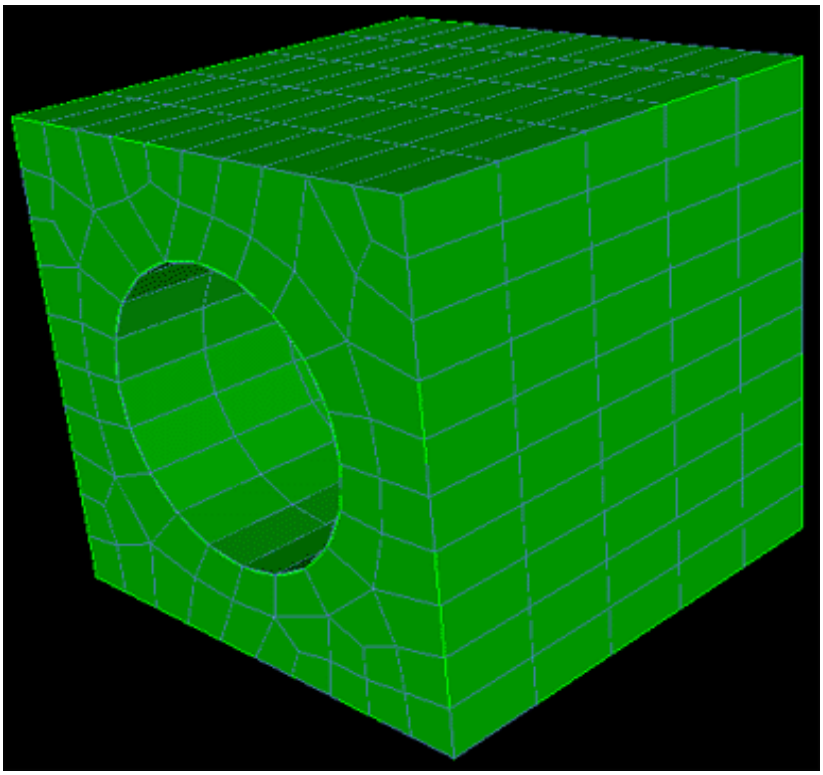
The type, quality, and speed of rendering the image can be controlled in CUBIT by selecting one of the buttons in the **Display** icon group. These icons appear by default in the icon bar above the graphics window. They can be used to change the display mode from wire frame, to hidden line or smooth shade.



For example, the following two figures result from selecting the **Hidden Line** and **Solid Smooth Shaded Mode** buttons respectively. Click here for more information on the Display icon group.



Hidden Line View of Mesh



Smooth Shade View of Mesh

Although CUBIT automatically computes limited quality metrics after generating a mesh and warns the user about certain cases of bad quality, it is still a good idea to inspect a broader set of quality measures. To do this, use the **Command Window** to enter the command:

CUBIT> quality volume 1 Allmetrics

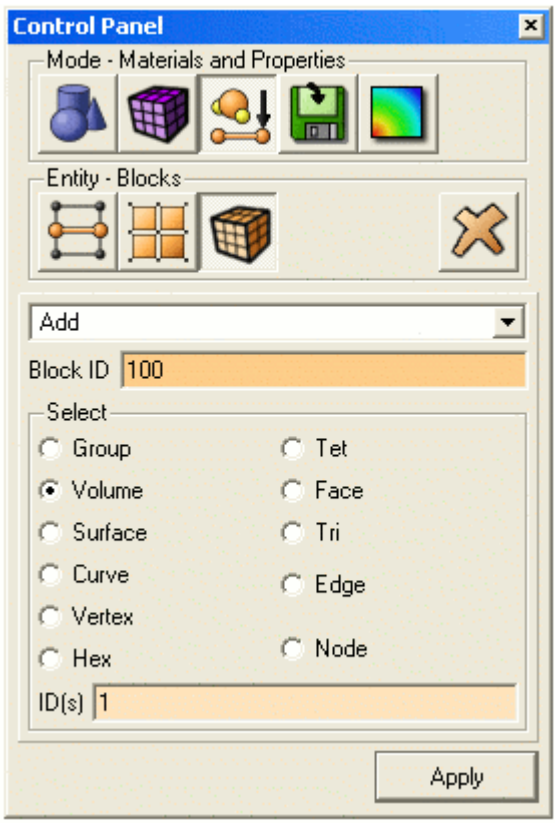
The results of the quality are displayed in the Command Window. For an explanation of each quality metric along with acceptable ranges, see Mesh Quality Assessment. For the purposes of this tutorial, you can assume the quality metrics shown are in an acceptable range.

GUI Basic Tutorial

Step 10: Defining Boundary Conditions

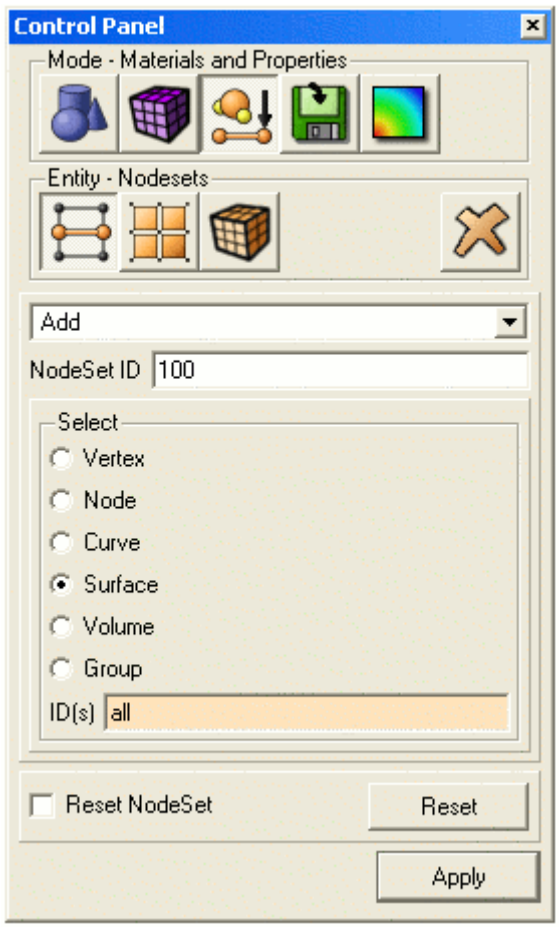
Let us assume that we need to define one material type for the entire mesh, and a single node-based boundary condition on all surfaces. This accomplished by identifying an Element Block and a Nodeset, respectively; the id numbers assigned to these entities are assigned by the user, usually by some convention meaningful to the analysis to be done. The element block and nodeset are identified from the Materials and Properties button on the control panel.

- Select the **Materials and Properties** button and then **Blocks** in the Control Panel window
- Select **Add** in the combo box
- Enter a **Block ID** of **100**
- Select the **Volume** radio button
- Enter the id of **Volume 1** by selecting it in the graphics window, or just manually entering in **ID(s)** field
- Press **Apply**



Create a nodeset by following the steps below

- Open the **Nodeset** window on the Control Panel
- Select **Add** from the combo box
- Enter a Nodeset id of **100**
- Select the **Surface** radio button and type all in the ID(s) field
- Press **Apply**

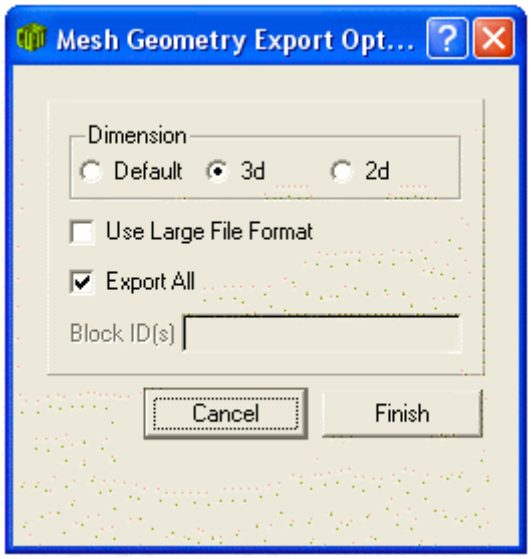


GUI Basic Tutorial

Step 11: Exporting the Mesh

Finally, the mesh needs to be written to an Exodus II file. This is easily done:

- From the **File** menu, select **Export**.
- Set the file export type to **Genesis Files** from file type combo box.
- Enter a file name in the dialog, such as **brick_with_hole.g**, and select **Save**. Since this is a standard file management dialog, the user may browse or use any other file management functionality supported by the platform.
- Set the dimension to **3d**
- Select the **Export All** check box



- Select **Finish** to export the mesh.

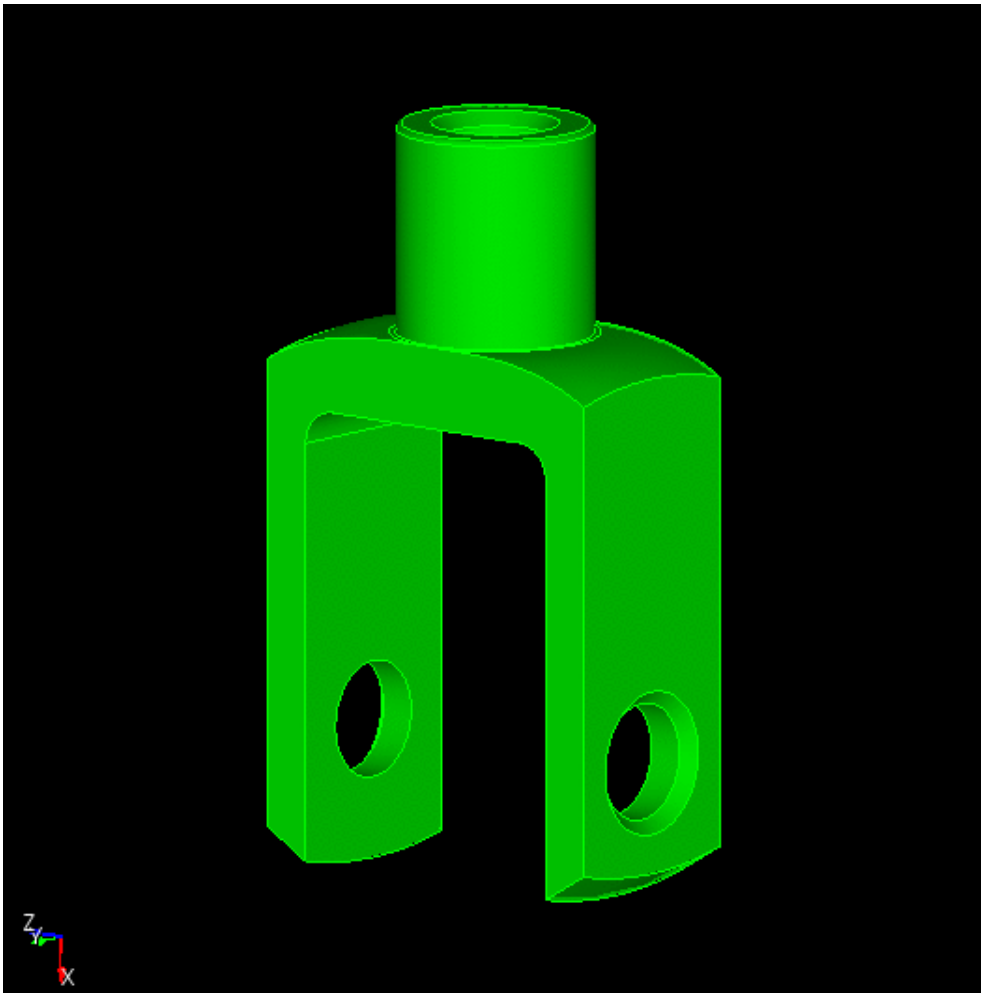
Power Tools GUI Tutorial

Overview

This tutorial demonstrates using the Power Tools on the CUBIT GUI for geometry decomposition and cleanup. The following features will be covered:

- Importing Geometry
- Analyzing Geometry
- Geometry Power Tools
- Webcutting
- Imprint/Merge
- Mesh Power Tools
- Meshing

Each of these steps is described in detail in the following sections. For this tutorial you will need to have a basic understanding of the CUBIT GUI functionality, including how to select entities, maneuver in the graphics window, operate the Control Panel, and use toolbars. If you have not already done so, we recommend completing the Basic Tutorial first. The following image shows the geometry that will be used for this tutorial.

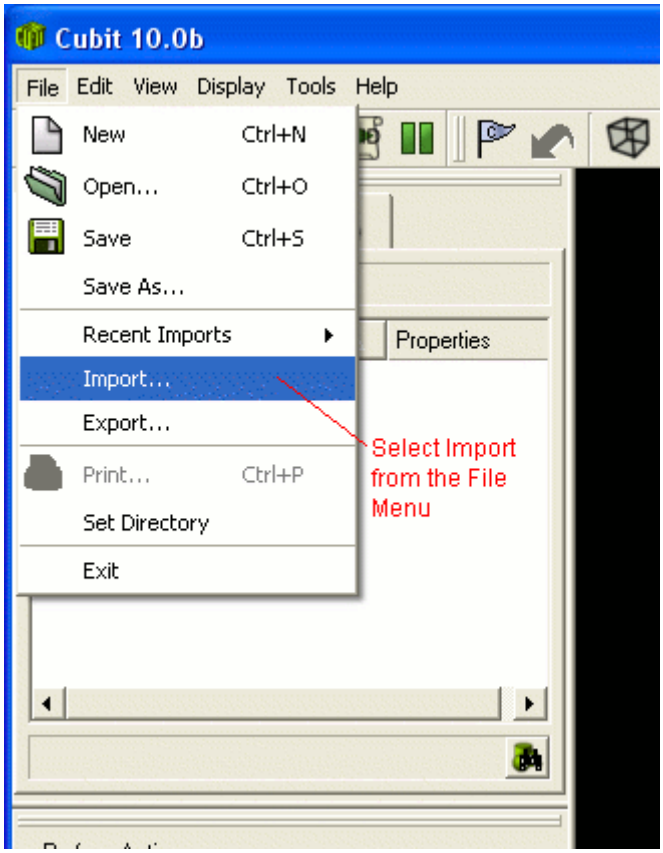


Power Tools GUI Tutorial

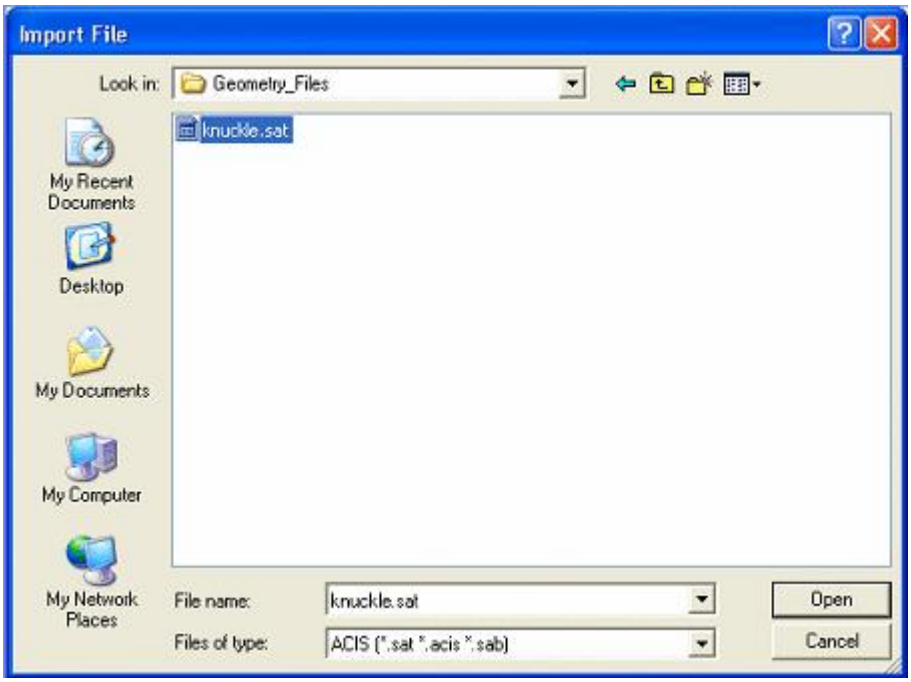
Step 1: Import the Geometry

Begin by opening a new session of CUBIT. To complete this tutorial, you will need to download the ACIS file that contains the geometry definition.

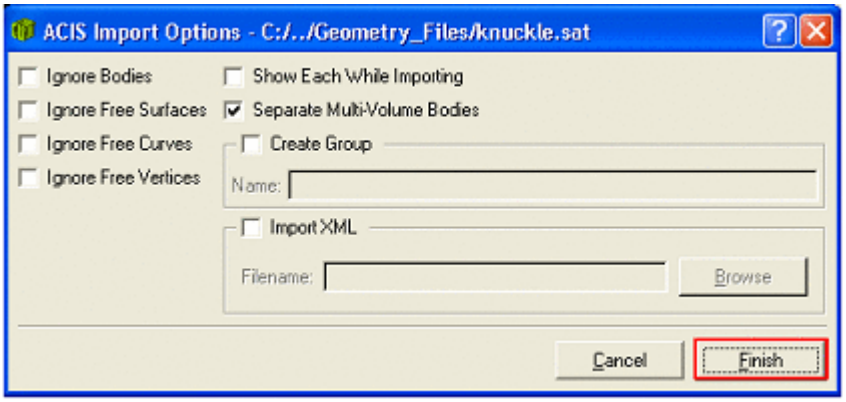
- Download geometry file knuckle.sat (Windows users can right click on the link and select "Save Target As")
- Select the **Import** option from **File** menu



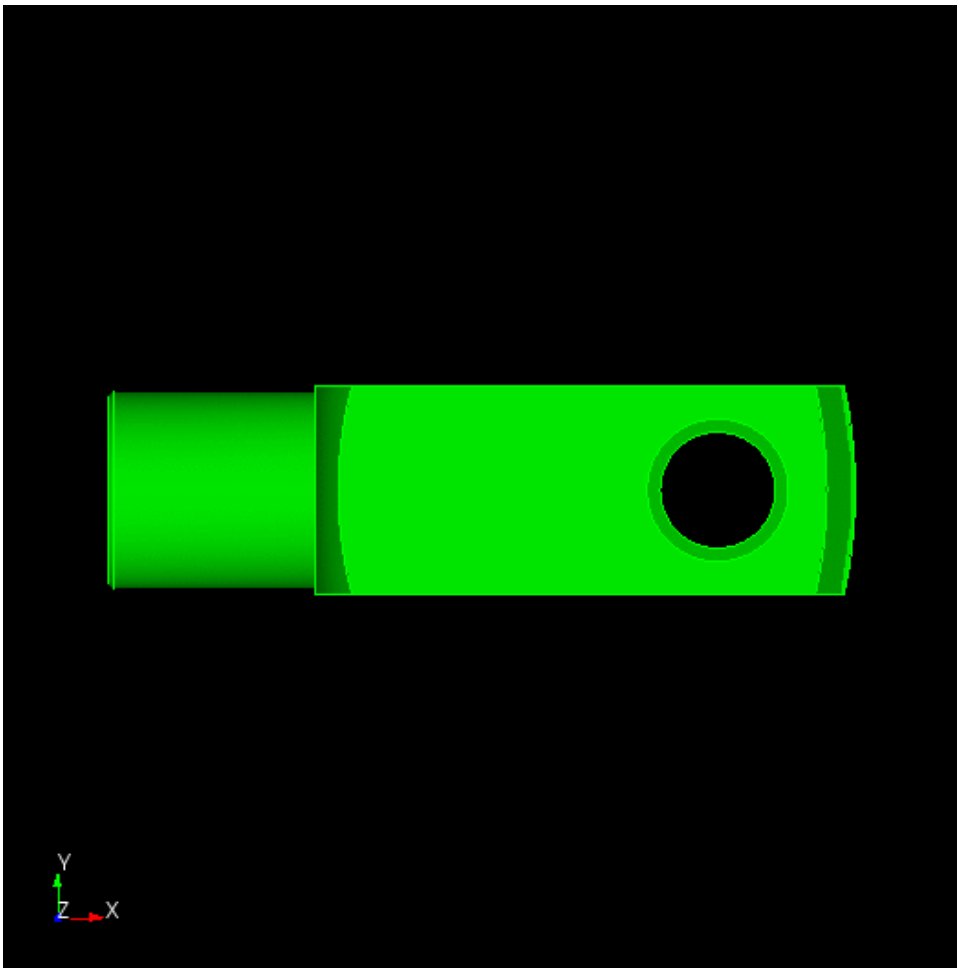
- The following dialog box will appear. Open the file by clicking on the name and selecting **Open**. If you do not see the file, make sure that you are in the right directory, and that the file type is set to ACIS.



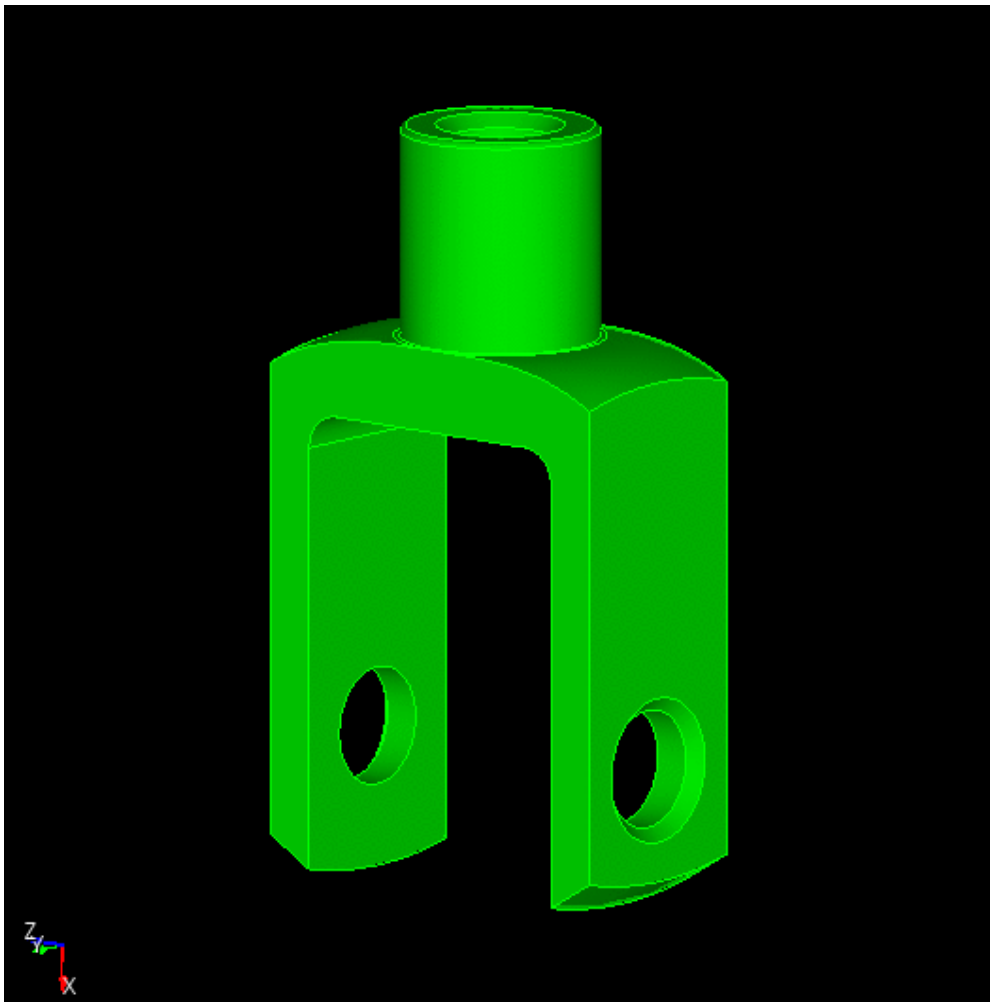
- Leave all of the import settings on their default settings and select **Finish**



Your graphics window should now appear as follows:



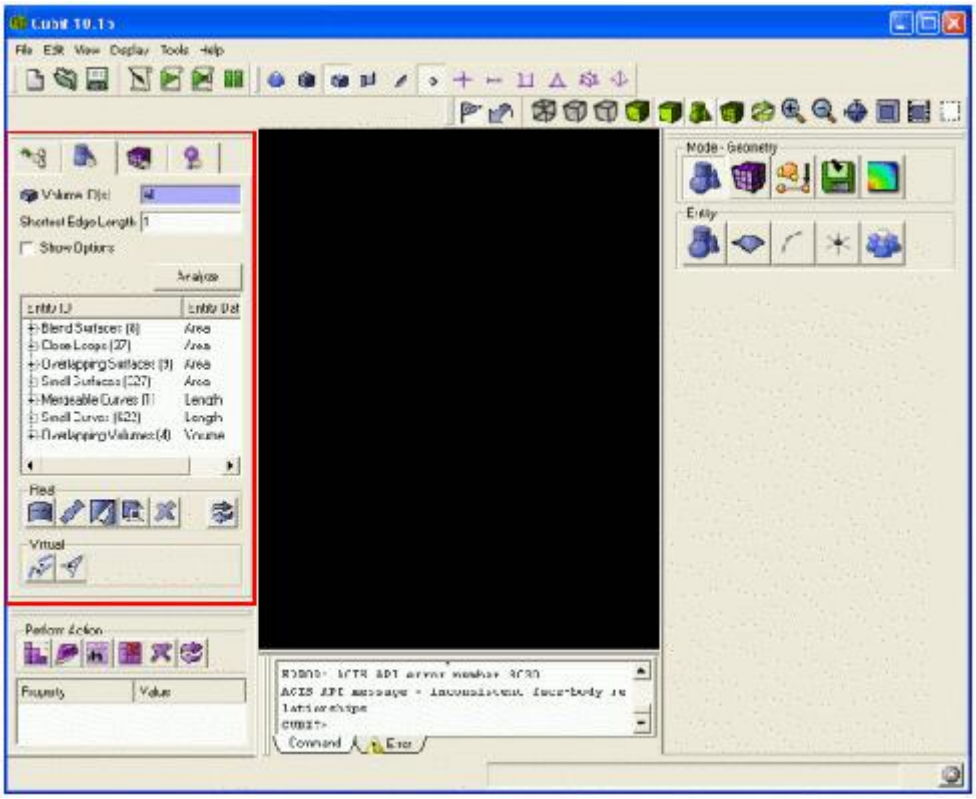
- Use the mouse to rotate the image in the graphics window to get a better perspective. For help with using the mouse in the graphics window, see Mouse Based Zoom, Pan and Rotate .



Power Tools GUI Tutorial

Step 2: Analyze the Geometry

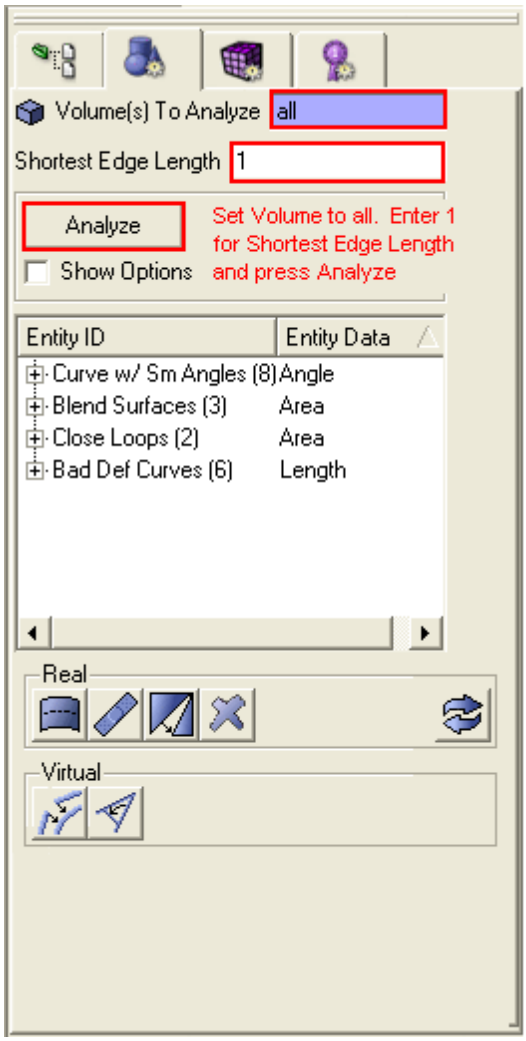
The Geometry Power Tools are located in the Entity Tree Window under the blue geometry tab. This menu provides access to many of the geometry analysis and clean-up tools in CUBIT.



Many geometries that are imported from other solid modeling software contain inconsistencies or small gaps that can cause meshing to fail. These problems are the result of differences in tolerances, file transfer loss, or inherent limitations in the parent system. In other instances, the geometry has no inconsistencies, but may be unsuitable for meshing because of topology such as small angles, overlap, or features smaller than the desired meshing size. The geometry analysis tool will analyze the volumes and return a list of suspected problems. To see a list of analysis options, click the "Show Options" box below the Analyze button.

Many of these problems can be fixed using the tools on the Power Tools menu. These include Split Surface, Heal, Tweak, Remove, Composite, Collapse. Most of these tools will be demonstrated in this tutorial.

- Open the **geometry repair** tab in the Entity Tree window
- Type all in the **Volumes to Analyze** field
- Set the **Shortest Edge Length** to 1
- Press **Analyze**



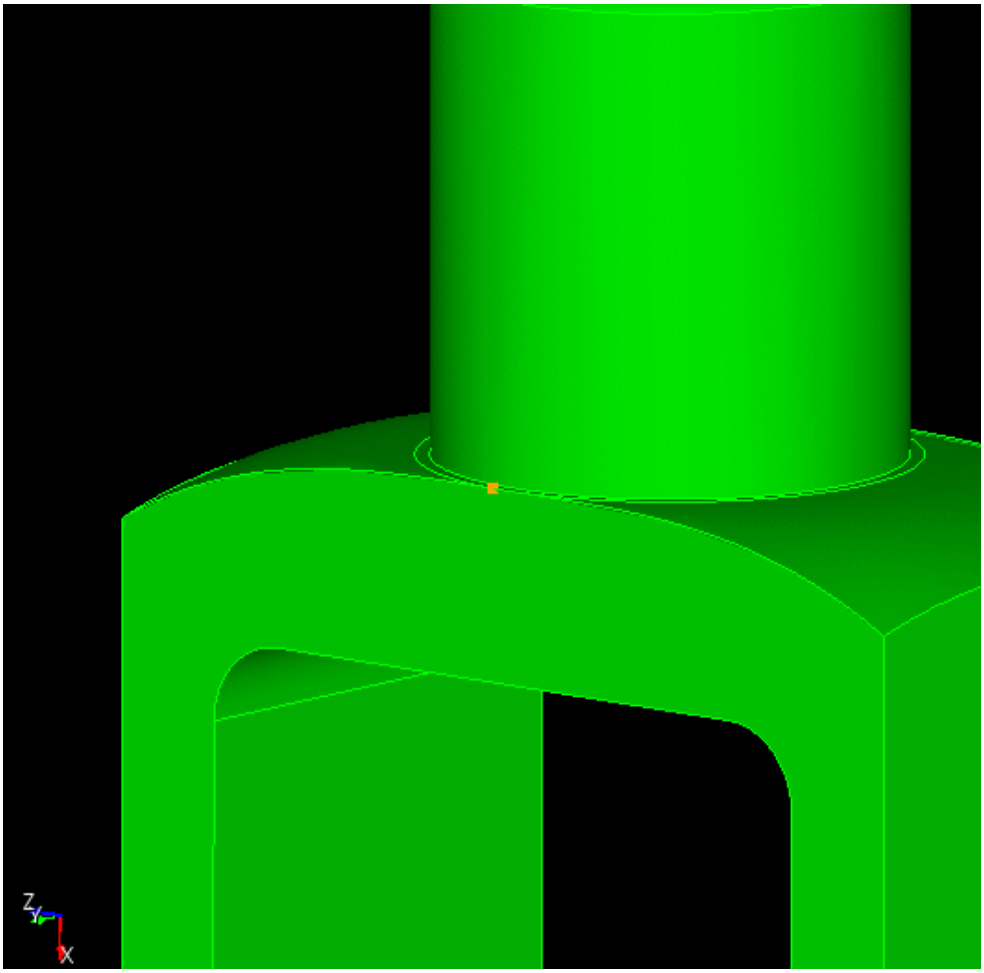
After the Analyze Button is pushed, display area will appear as shown above. There are four suspected problems with this geometry: Curves with Small Angles, Blend Surfaces, Close Loops, and Badly Defined Geometry. The numbers in parentheses indicate the number of occurrences of this problem in the model. Clicking on the + sign by each label will list the CUBIT entities by ID with this problem. Clicking on the + sign by each entity will cause that entity's children or parents to be listed (depending on the entity and the type of geometry test). See documentation on Geometry Repair for more information about the display window. Clicking on the name of an entity will highlight that entity in the graphics window.

- Select **Vertex 45** under Curves with Small Angles

Observe that this vertex is highlighted in the graphics window.

- Right click and select **Zoom To** from the list of options

The graphics window should look like this:



- Right Click on Vertex 45 and select **Reset Zoom** from the list of options

The image should now be reset to the previous graphics state.

You can experiment with some of the other options in the top half of the right click menu. They are:

- **Fly-in** - Animated zoom feature
- **Locate** - Labels entity
- **Draw** - Draw this entity by itself
- **Draw with Neighbors** - Draw this entity with all adjacent curves and surfaces
- **Clear Highlights** - Clear all highlighted entities
- **Reset Graphics** - Refresh graphics screen

The graphics window may also be reset by pressing the reset graphics button on the menu.

Power Tools GUI Tutorial

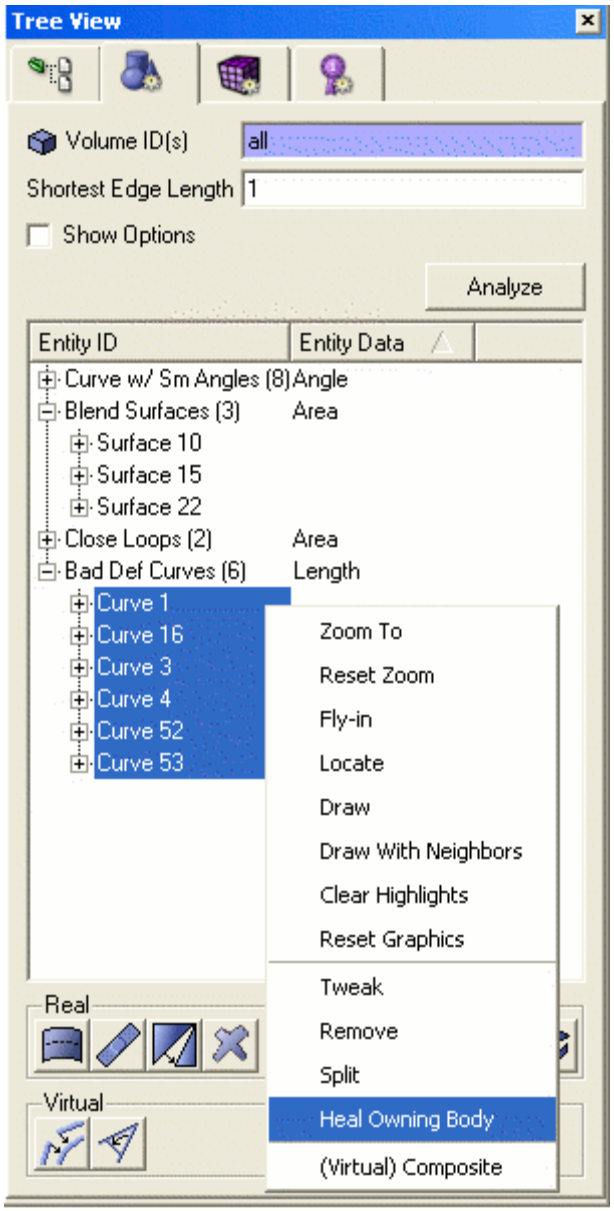
Step 3: Healing the Geometry

The first step to improving any geometry is to look for badly defined geometry and to fix it using the Autoheal tool in CUBIT. The Geometry Analysis tool may detect these inconsistencies, but only if such a function exists in the parent software. It is always a good idea to run the Autoheal on imported geometry. In this example, the Power Tools has located some badly defined curves. This step will show you how to use the geometry repair tool to fix these curves.

- **Highlight** all of the badly defined curves by holding down the Shift key while selecting
- Right click and select **Heal Owning Body** from the list of options

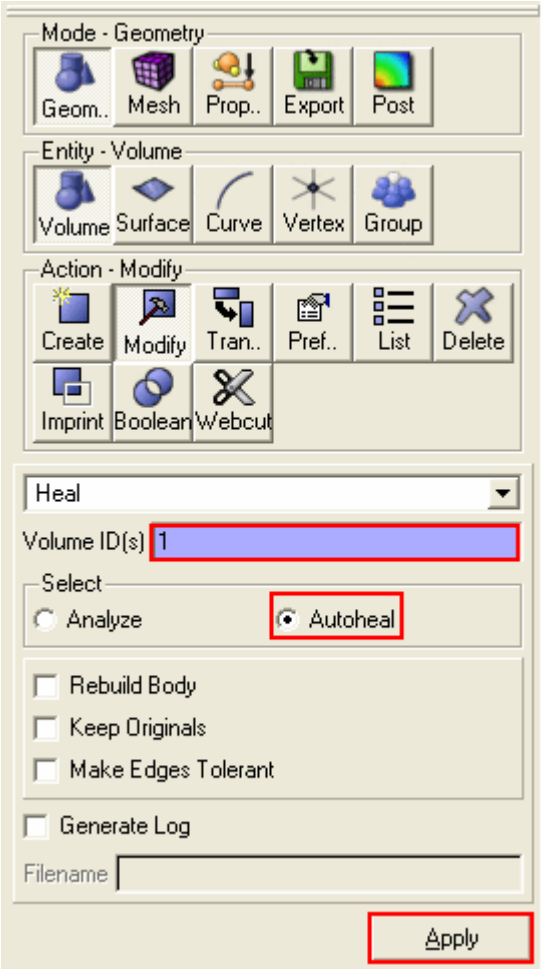
OR

- Click the button



The Geometry Repair Tool does not execute any geometry clean-up commands directly, but directs you to the place on the Control Panel where this function can be executed. The following menu will appear on the Control Panel. Notice that the id of the owning body has already been pasted into the input window.

- Select the **Autoheal** button
- Press **Apply**



The output window on the CUBIT GUI should appear with the following message. You may have to scroll to see the whole thing. The percentage before and after healing are 67% to 100%. Healing has been successful.

```
CUBIT> healer autoheal body 1
Healing Volume 1...
  Percentage good geometry before healing: 67%
  Percentage good geometry after healing:  100%
 Successfully healed Volume 1!
Updated volume(s): 1
Modified volume 1

Percentage good geometry in Volume 1: 100%

HEALER ANALYSIS SUMMARY:
------------------------
Analyzed 1 Volume: 1
Found 0 bad Vertices.
Found 0 bad Curves.
Found 0 bad CoEdges.
Found 0 Volumes with problems.
Journaled Command: healer autoheal body 1

CUBIT>
```

Script | Command | Error

Run the geometry analysis test again to guarantee that all bad geometry has been removed.

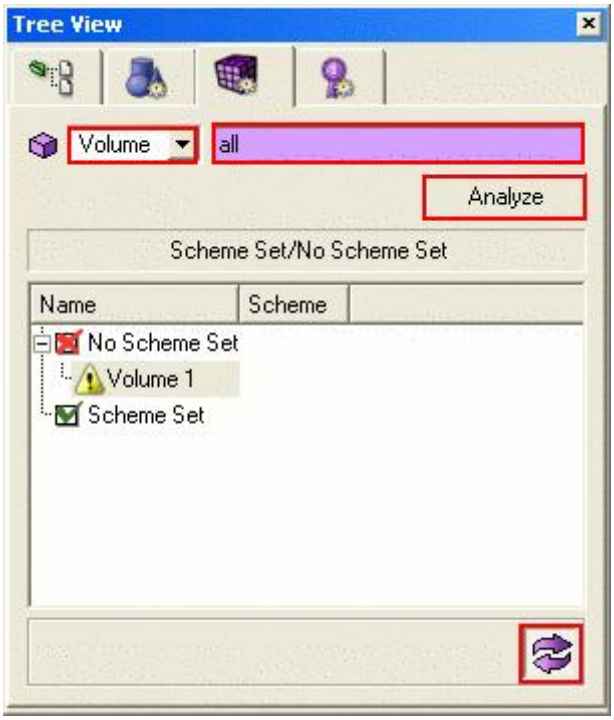
- Press the **Analyze** Button in the Geometry Repair window

Power Tools GUI Tutorial

Step 4: Mesh Power Tools

The Mesh Power Tool provides an easy and graphical way to determine if volumes are meshable. This tool will employ the AutoScheme feature in CUBIT to select and assign schemes to meshable volumes. If a volume is not currently meshable, it will be flagged and highlighted. Use the Mesh Power Tool to determine if the volume is currently meshable.

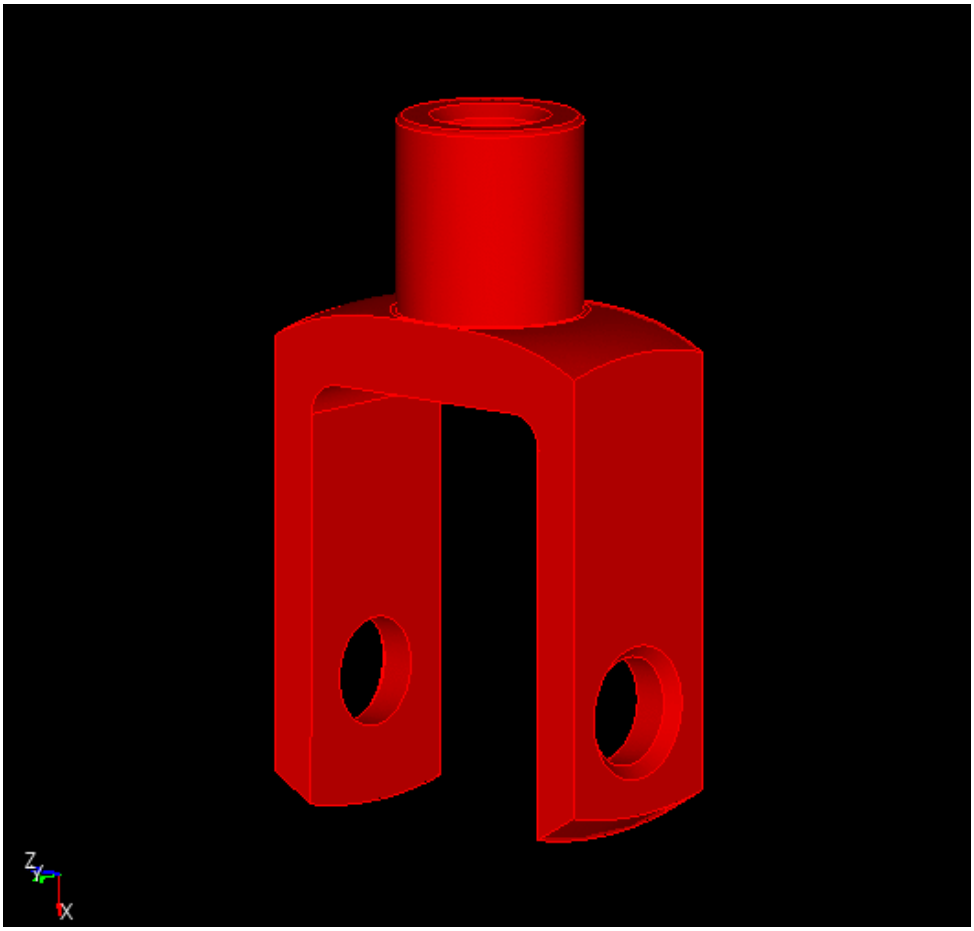
- Click on the purple Mesh Tools tab in the Power Tools window.
- Select **Volume** as the entity type in the pulldown menu (It may already be selected)
- Enter **all** in the input window
- Press **Apply**



Volume 1 will appear under the "No Scheme Set" heading.

- Toggle the **Graphics Button** in the bottom left corner

The graphics window should look like this with Volume 1 highlighted in red. Using this graphics feature, all volumes that are meshable will be highlighted in green, and all volumes that are not currently meshable will be highlighted in red.



- Turn the **Graphics Button** off so that Volume 1 is shown in green again

Power Tools GUI Tutorial

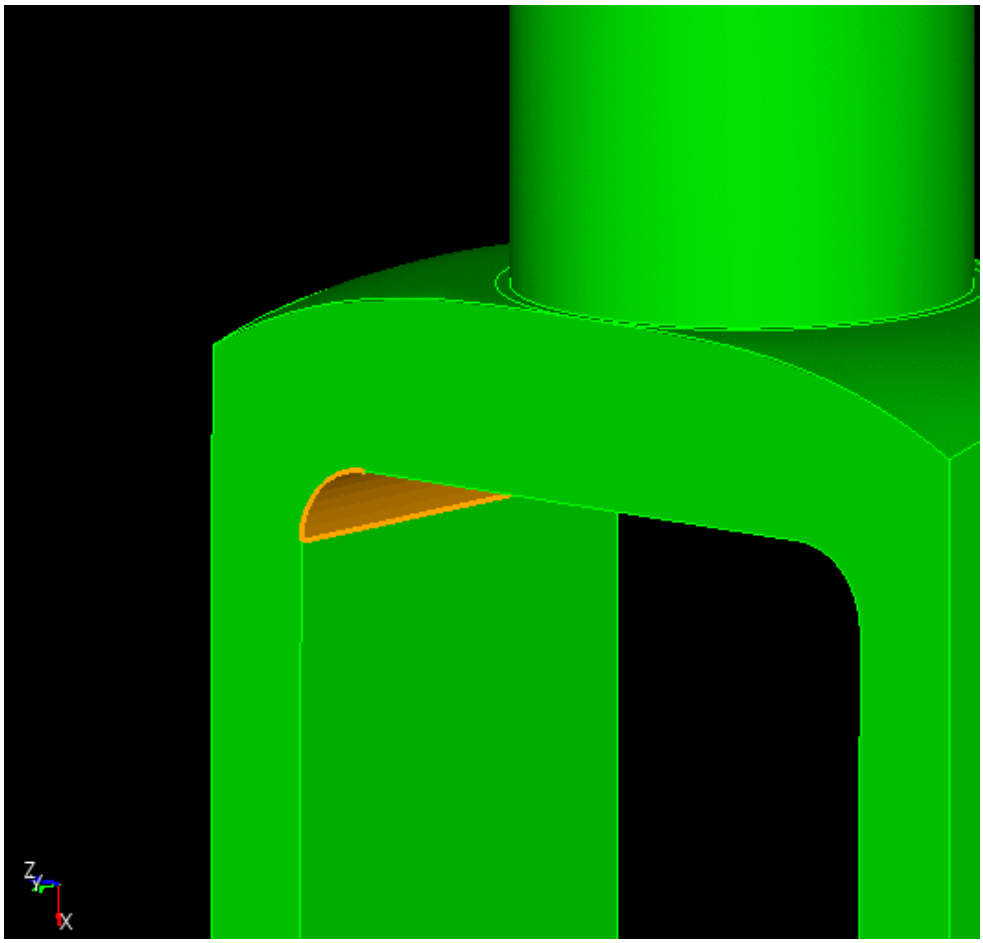
Step 5: Splitting Filleted Surfaces

The previous step determined that the volume was not currently meshable, and that further decomposition was required. This decomposition can be performed using the tools in the Geometry Repair power tools. A good place to start is with blend surfaces.

A blend surface is a transitional surface that connects two orthogonal planes, also known as a fillet. Blend surfaces can be problematic in meshing because there is no clear transition between the two orthogonal surfaces, making sweeping or mapping algorithms difficult. The Split Surface function divides these blend surfaces (or any surface) into two distinct surfaces.

- Select **Surface 22** from the list of blend surfaces
- Right click and select **Zoom To** from the list of options

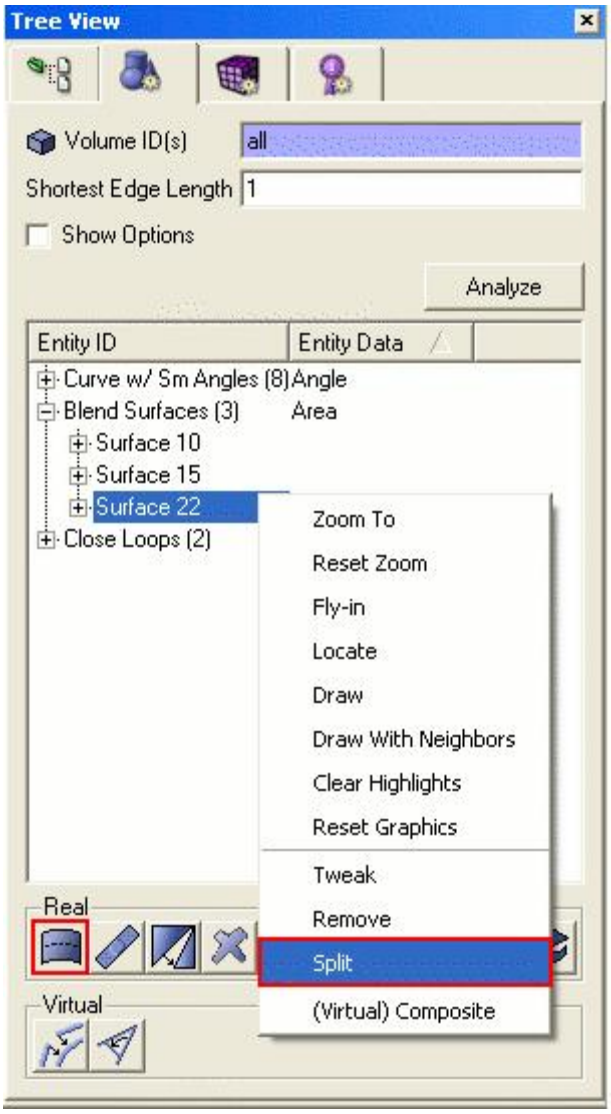
The graphics window should look like this:



- Right click with Surface 22 highlighted and select the **Split** button

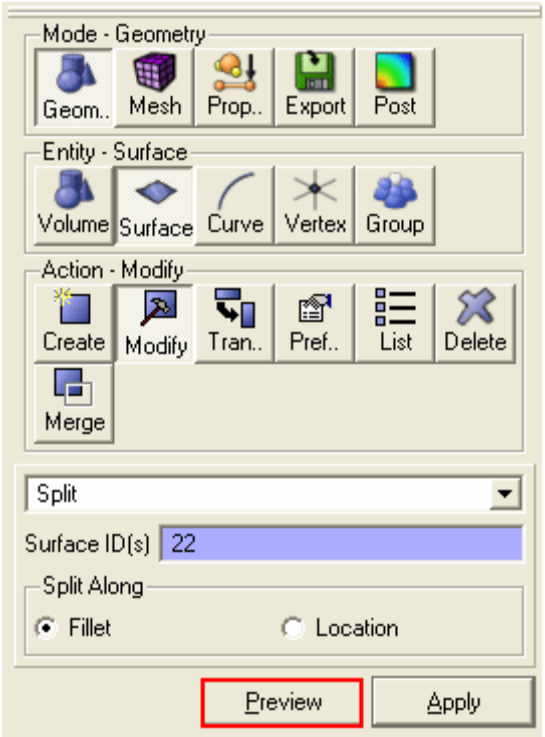
OR

- Click the button on the tool panel

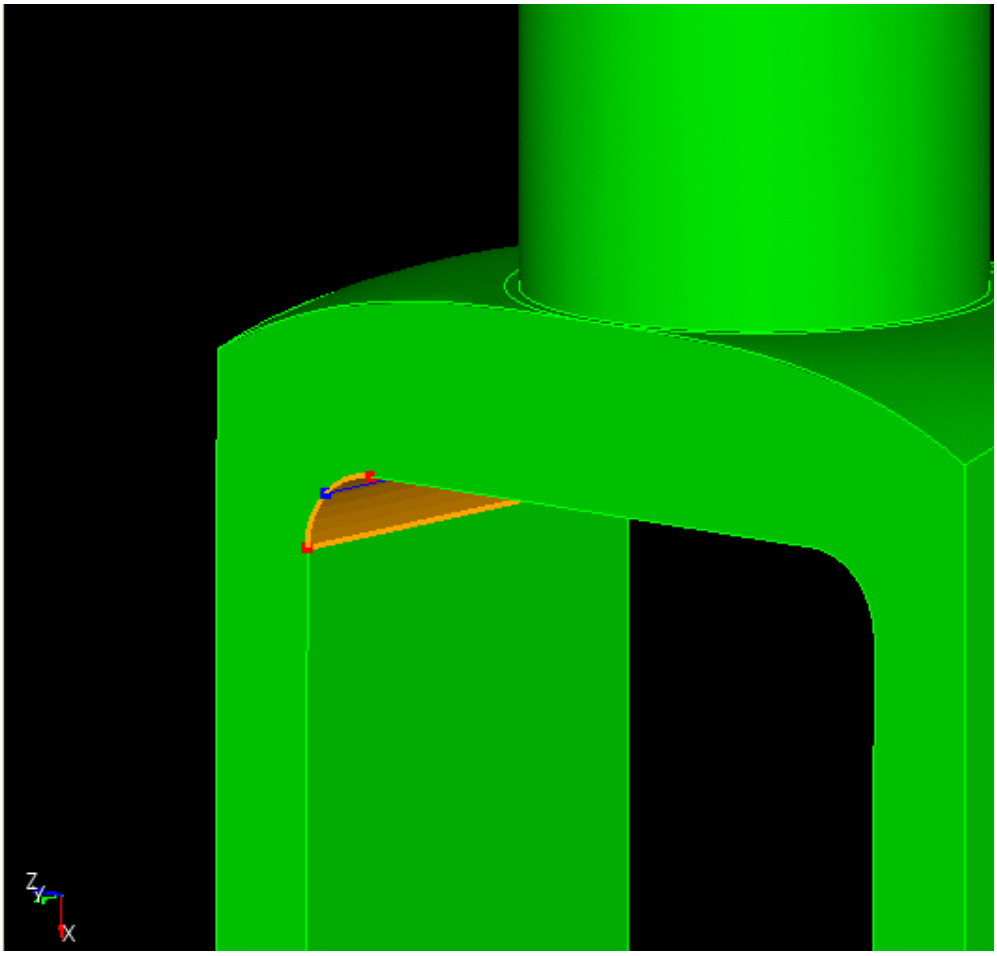


The Geometry-Surface-Modify-Split Menu will appear on the Control Panel. Make sure the Surface id is input in the window.

- Press the **Preview** Button



The blue line shows where the surface will be split.



- Press the **Apply** Button

The surface should now appear split.

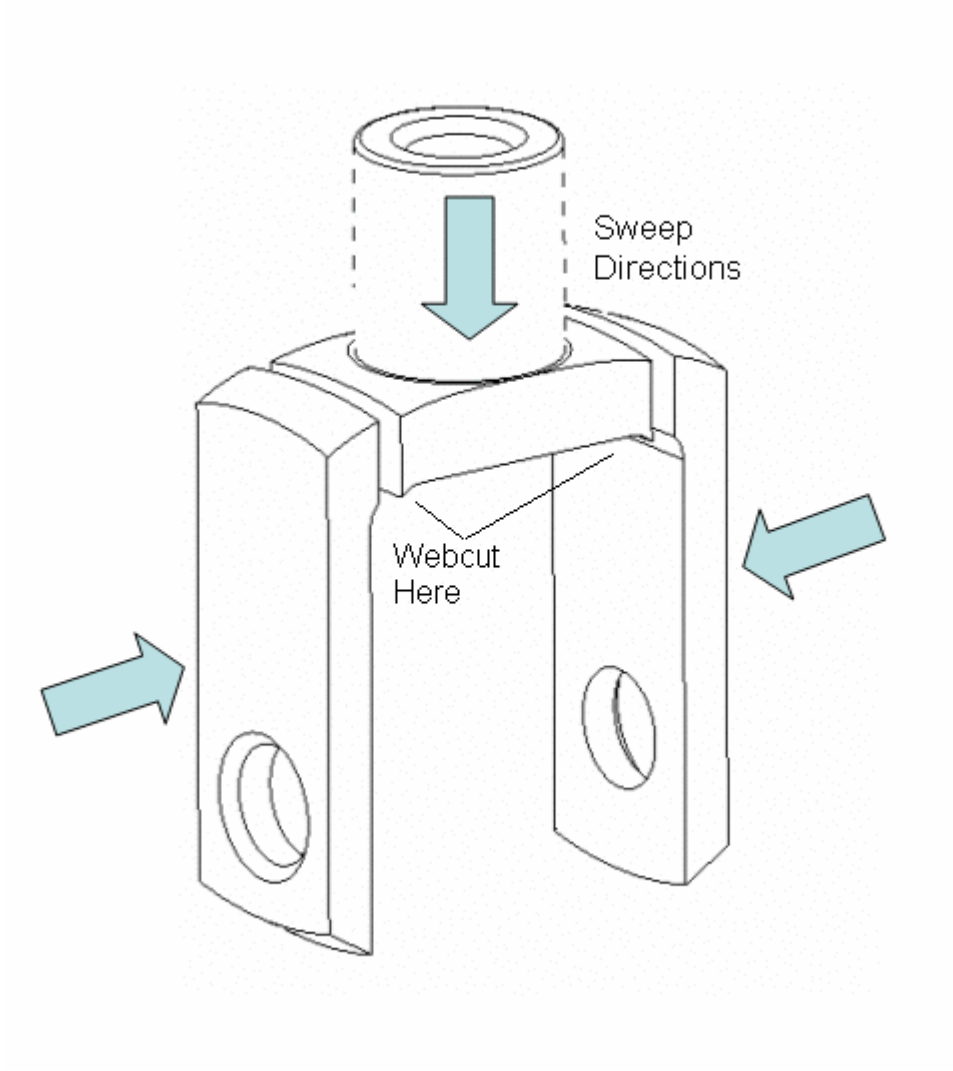
- Repeat these steps with the opposite blend surface (ID 10)

Power Tools GUI Tutorial

Step 6: Web Cutting

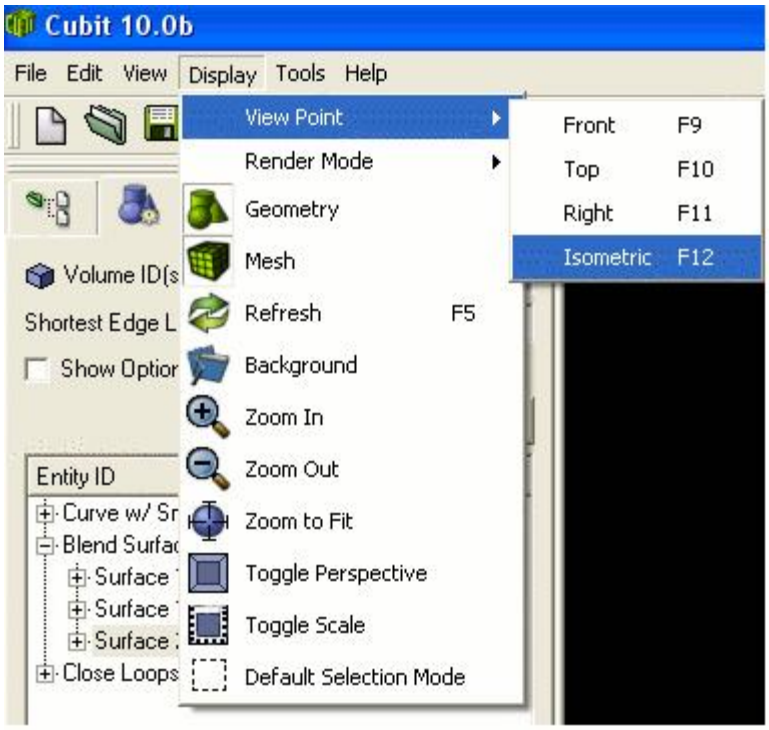
Since the model has several through holes, sweeping is not possible from a single source and target. However, it is possible to divide the model into three sweepable regions. The figure below shows where to divide the model to get it into sweepable regions. These regions coincide with the holes in the model.

Web cutting is this process of dividing volumes into sweepable regions by cutting with a plane. For this exercise, you will use the curves that were just created with the split surface command to cut the volume.



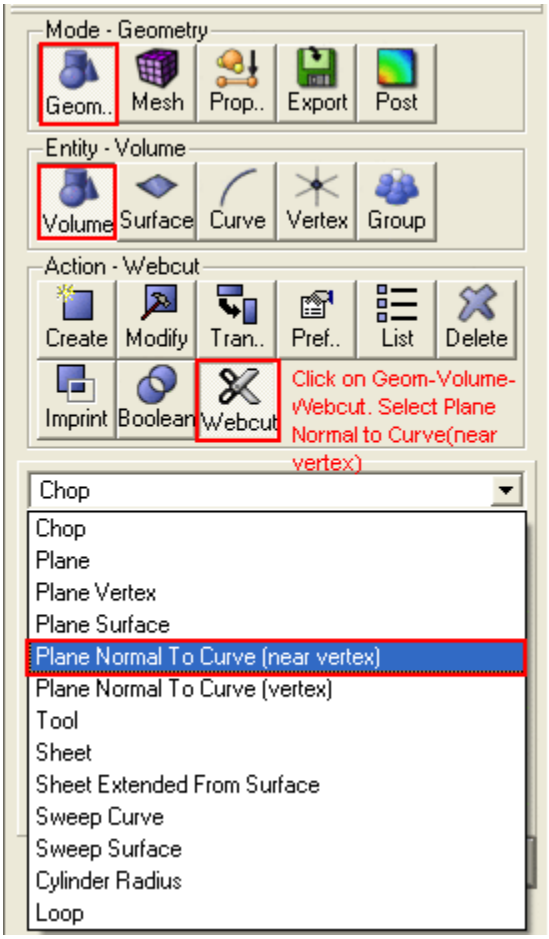
In order to visualize the process more clearly, switch to the isometric view.

- Change the view to isometric in the **Display** menu under **View Point**

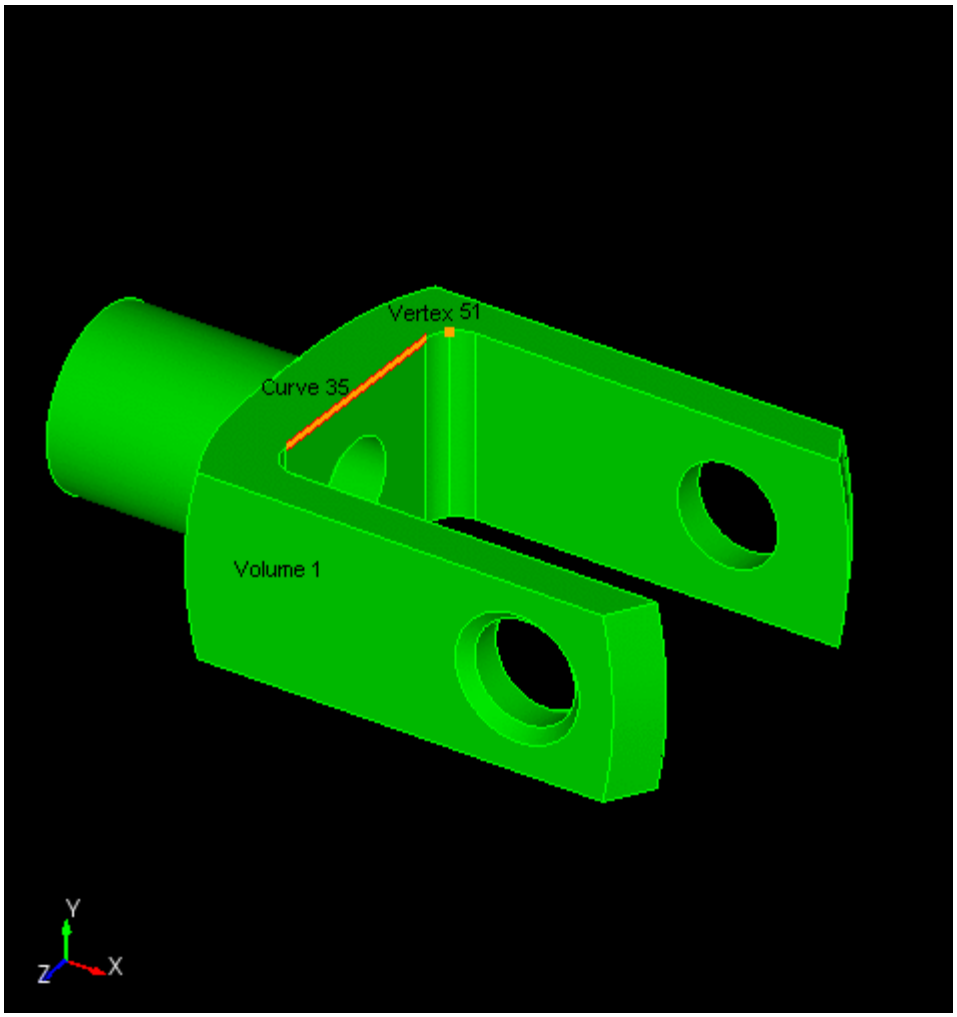


The web cutting menu is located under Geometry-Volume-Webcut on the Control Panel.

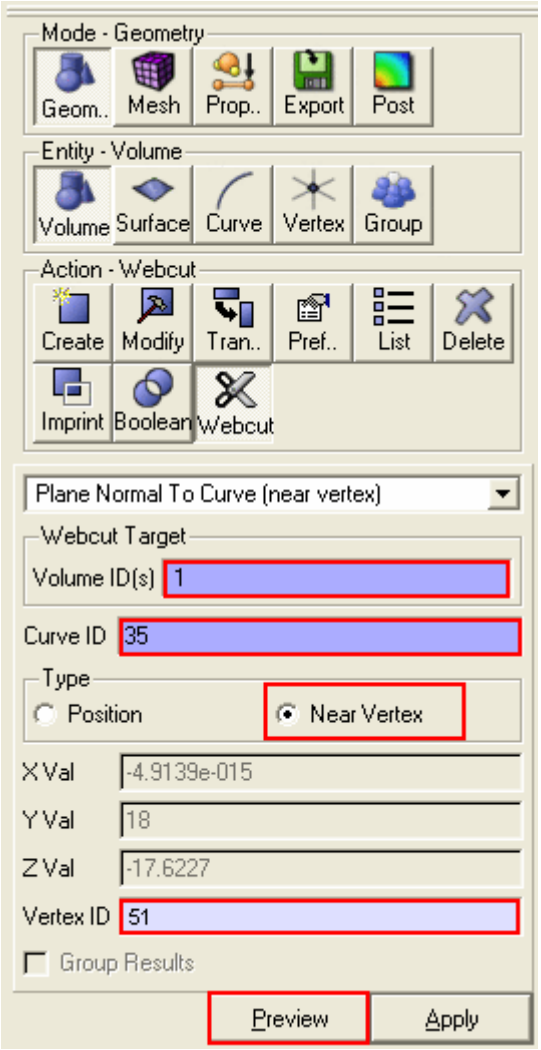
- Click on **Geometry**, then **Volume**, then **Webcut** on the Control Panel
- Select **Plane Normal to Curve (near vertex)** from the list of options



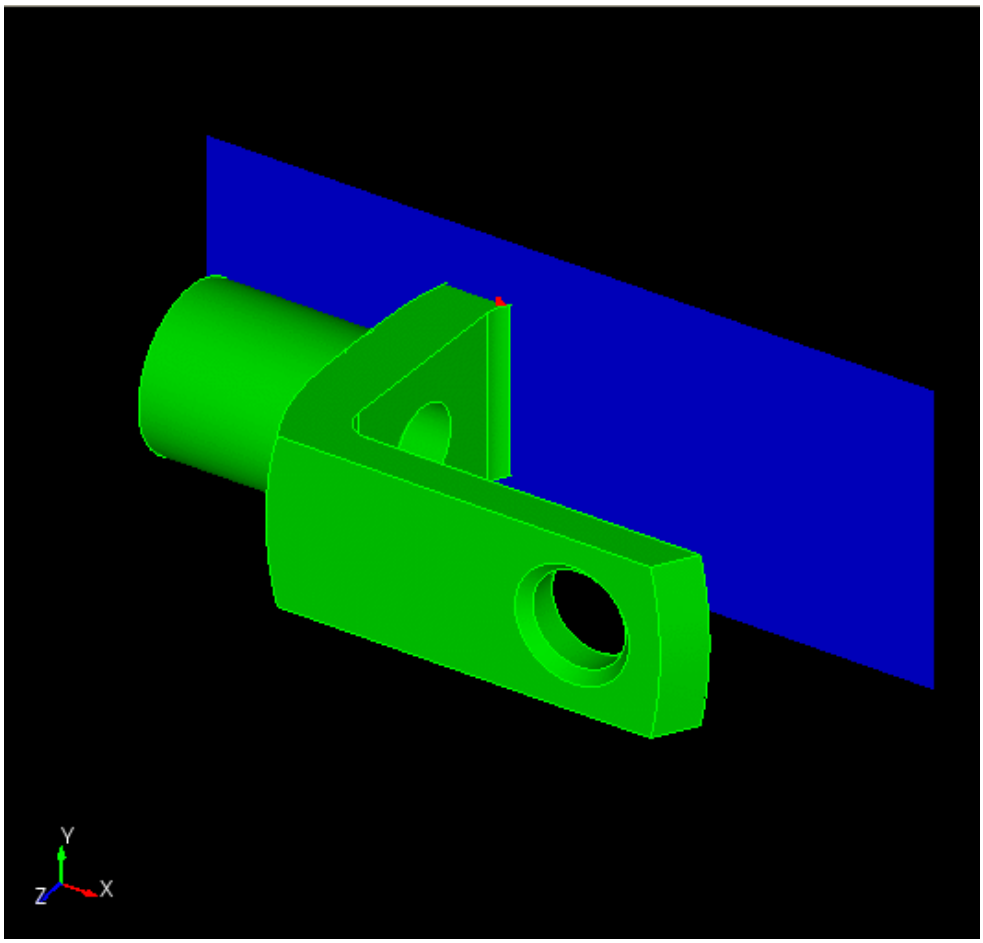
The following image shows the entity ids that will be used to webcut the volume. Select entities with the mouse by clicking on them.



- Enter **Volume 1** by typing it or selecting from the graphics window
- Enter **Curve 35** by typing it or selecting from the graphics window
- Change the **Type** to **Near Vertex**
- Enter **Vertex 51** by typing it or selecting from the graphics window
- Press **Preview**



A blue preview plane should appear in the following position. Check to make sure that your model looks the same.



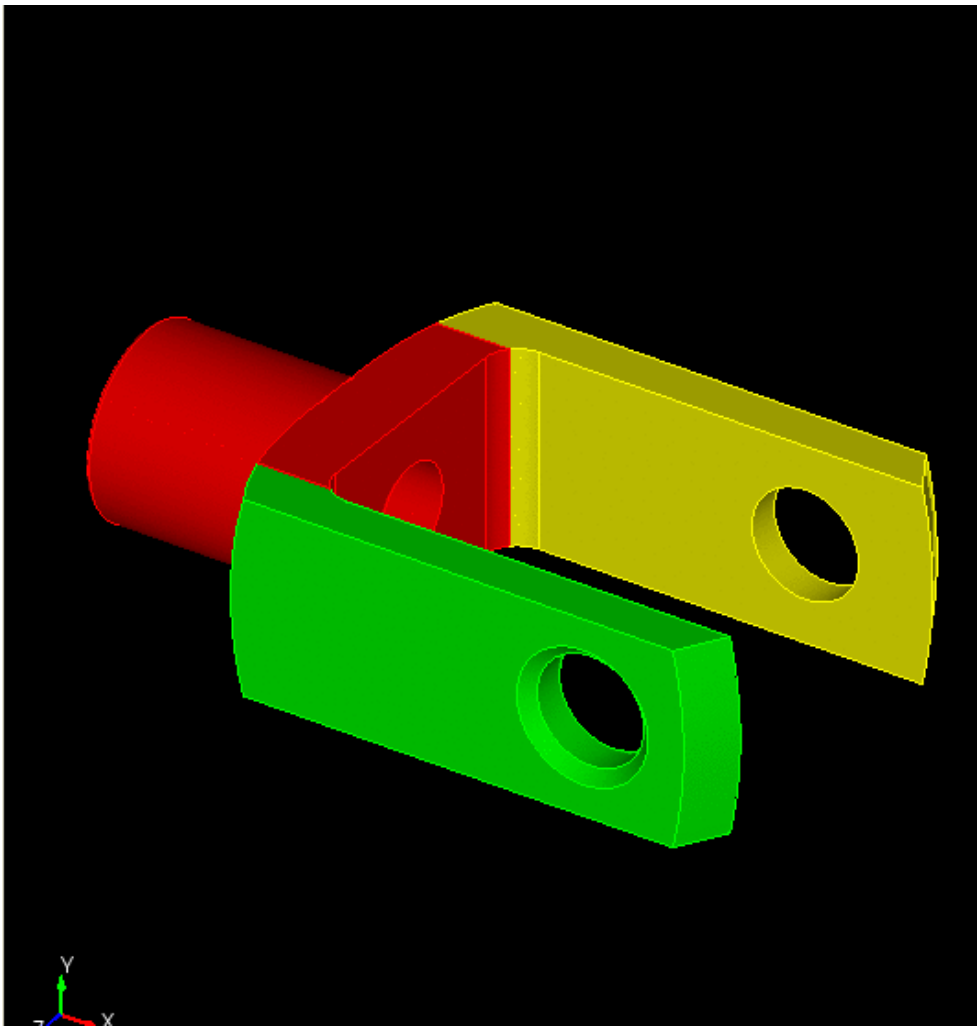
- Press **Apply**

The volume has now been split into two volumes. Volume 2 is shown in yellow.

Repeat these steps with the other side of the part. The Volume and Curve ids will remain the same.

- Enter **Vertex 49** in the input window or select from the graphics window
- Press **Preview**, then **Apply**

The final webcut volume should look like this:

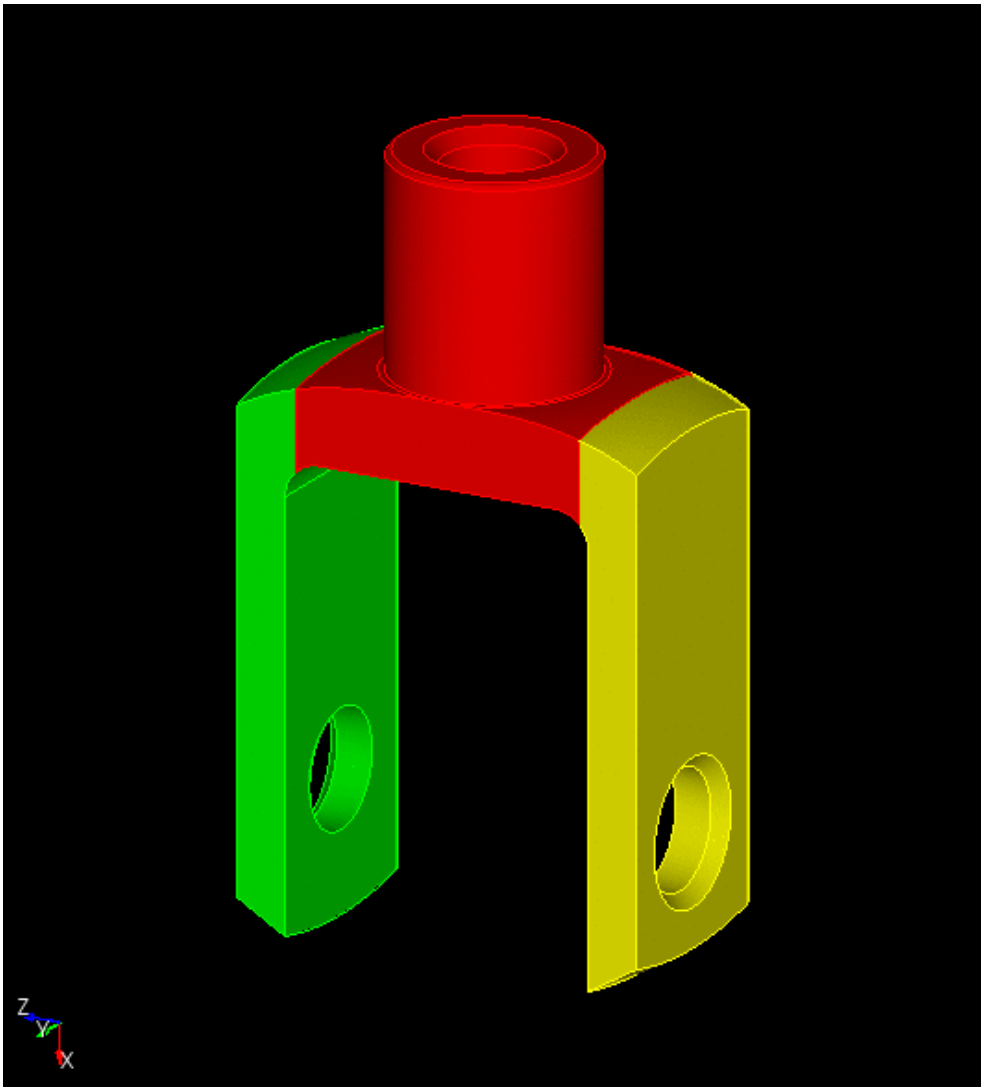


Power Tools GUI Tutorial

Step 7: Removing Small Surfaces

Some surfaces are too small for analysis and should be removed from the model. In this example, Surface 15 and Surface 17 may fall into that category, assuming that the distance between curves on these surfaces is smaller than the desired final mesh size. You can remove these surfaces by extending adjacent surfaces until they intersect.

- **Rotate** the model to the following orientation



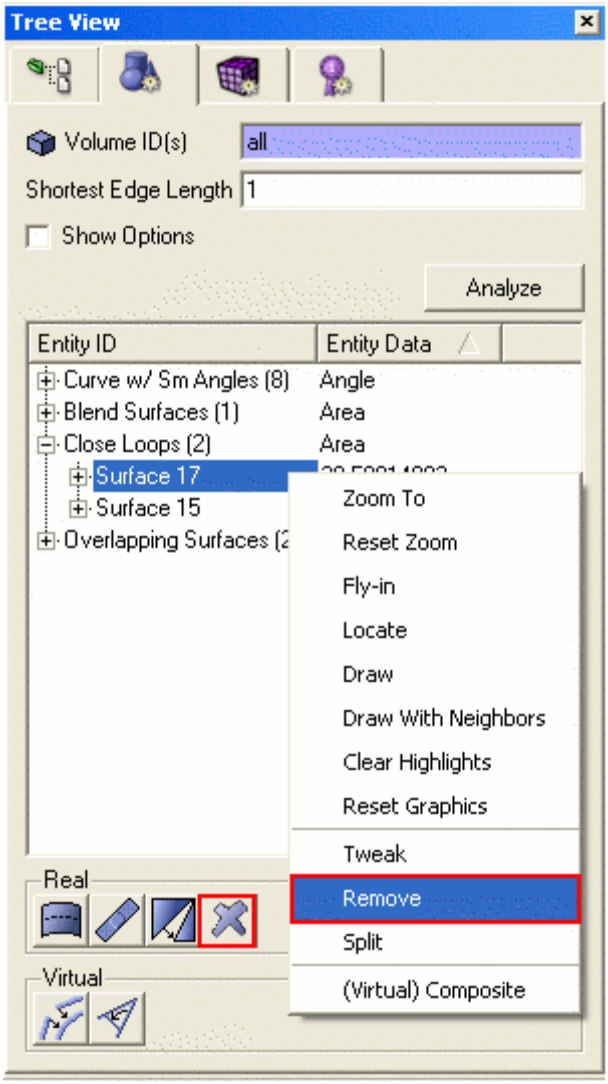
- Press **Analyze** on the Geometry Power Tools menu

You will notice that a new category has appeared labeled Overlapping Surfaces. This is because there are two new surfaces created for each of the webcuts that overlap a surface on the original body. This can be removed using the Imprint/Merge function which will be explained in Step 9.

- **Zoom** to **Surface 17** in the graphics display
- **Right Click** on **Surface 17** in the Geometry Repair window and select **Remove**

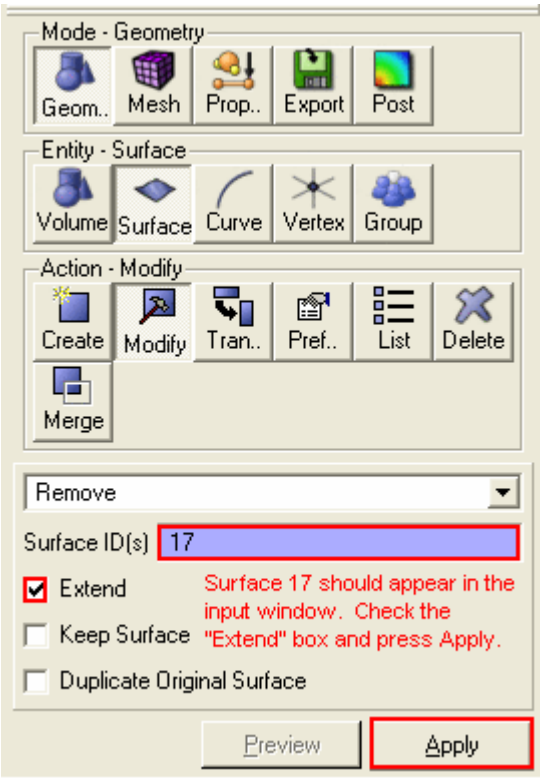
OR

- Press the **Remove Button** on the tool bar

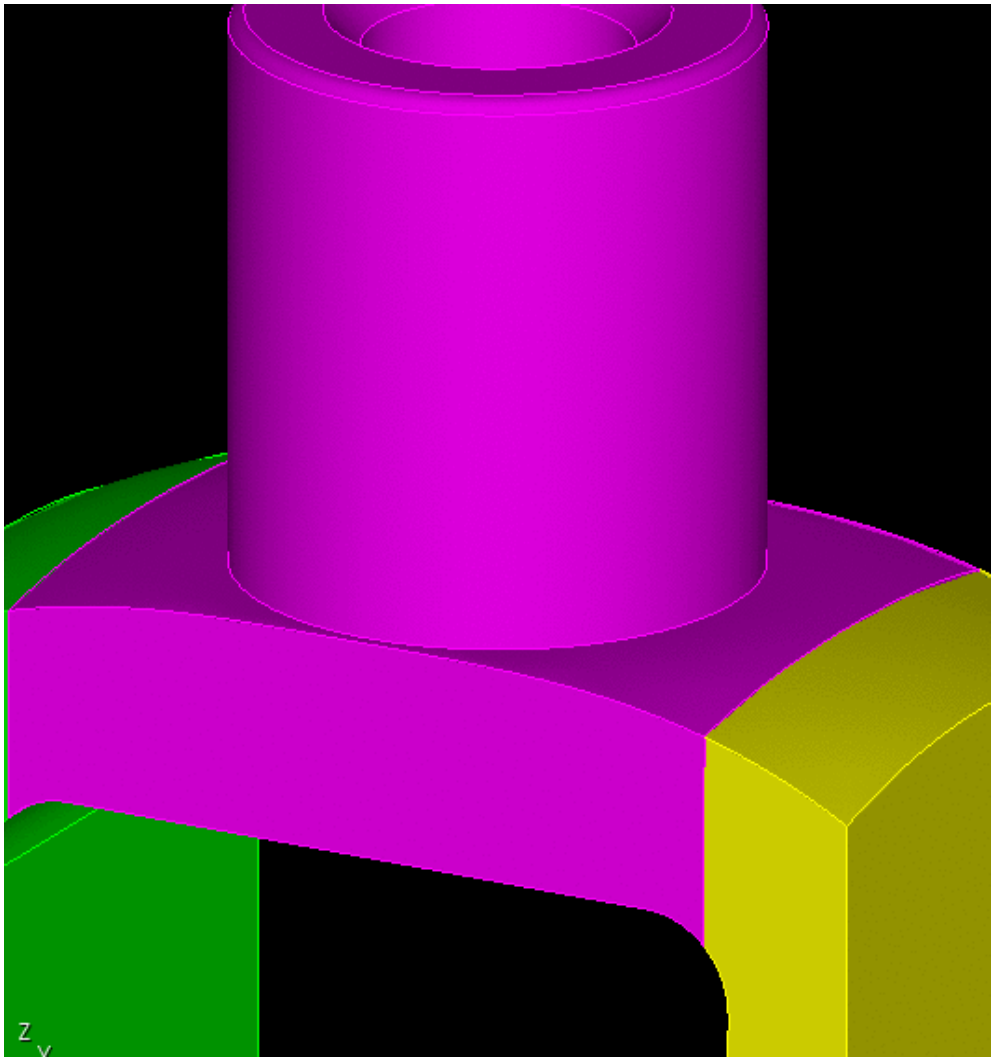


The Control Panel will appear under the Geometry-Surface-Modify- Remove heading. The Surface id should appear in the input window.

- Make sure that **Surface 17** appears in the window and the **Extend** button is checked
- Press **Apply**



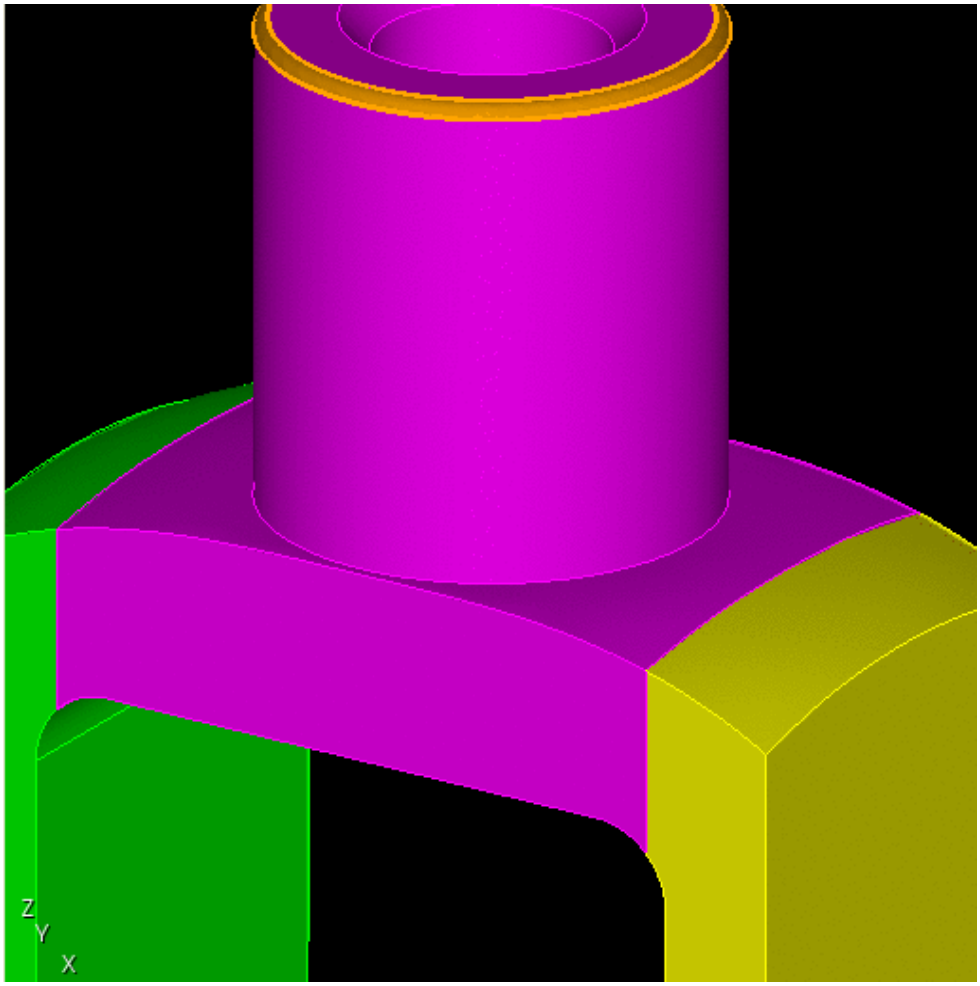
Two things of interest have changed in the graphics window. The small surface no longer appears, and the id of the volume has changed. The Remove command has in essence, rebuilt a new body from scratch to replace the old one. All of the Curves and Surface ids have also changed.



Since all of the surface IDs have changed you will need to run the Geometry Analysis again.

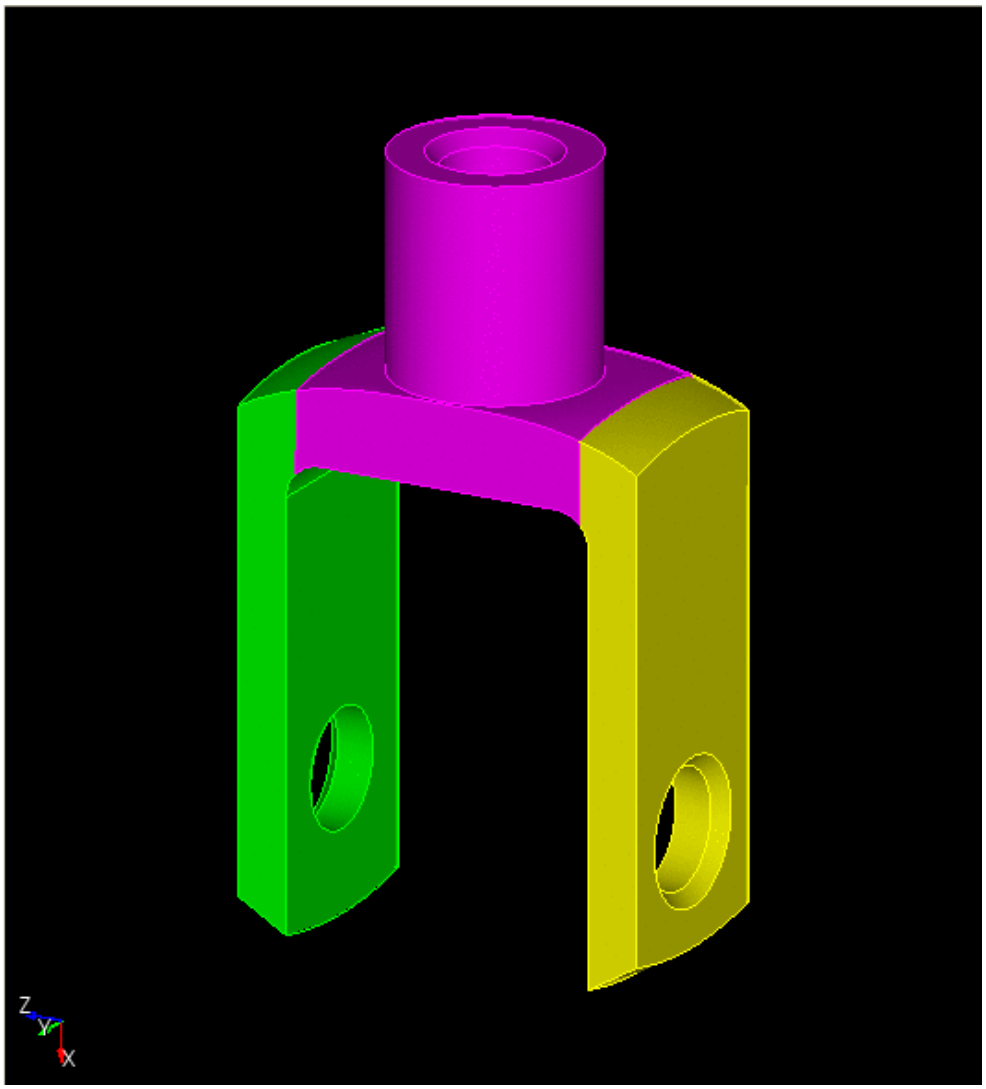
- Press **Analyze** on the Geometry Repair Tool
- Highlight **Surface 55** and select the **Remove** option

Surface 55 is shown highlighted in the following image.



- The Geometry-Surface-Modify-Remove option appears on the Control Panel. Make sure that **Surface 55** appears in the input window.
- Press **Apply**

This time removing the surface did not require a rebuild, so the body id remains the same. Reset the Zoom to show the entire model.



Power Tools GUI Tutorial

Step 8: Tweaking Surfaces

Tweaking is the process of deleting, moving, or offsetting, surfaces and extending or trimming adjacent surfaces to fill in the gaps. Tweaking is useful for eliminating gaps between components, simplifying geometry or changing the dimensions of an entity. Tweaking will be used in this example to decrease the radius of the upper cylinder.

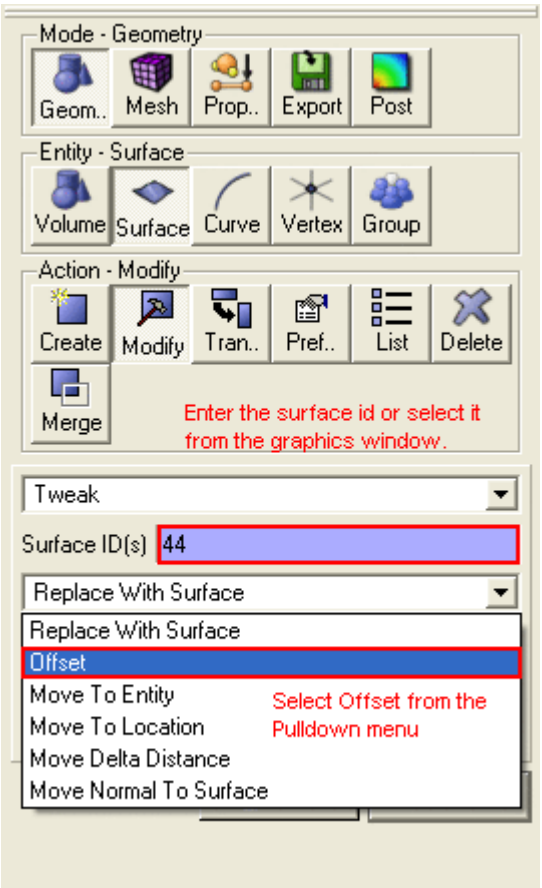
Begin by reanalyzing the geometry.

- Press **Analyze** on the Power Tools menu

There should be 1 entry under the "Close Loops" category for Surface 46. A close loop (pronounced KLOS) is a surface which has two loops that are within some small distance of each other at their closest points. The parameter for distance is the square of the shortest edge length parameter.

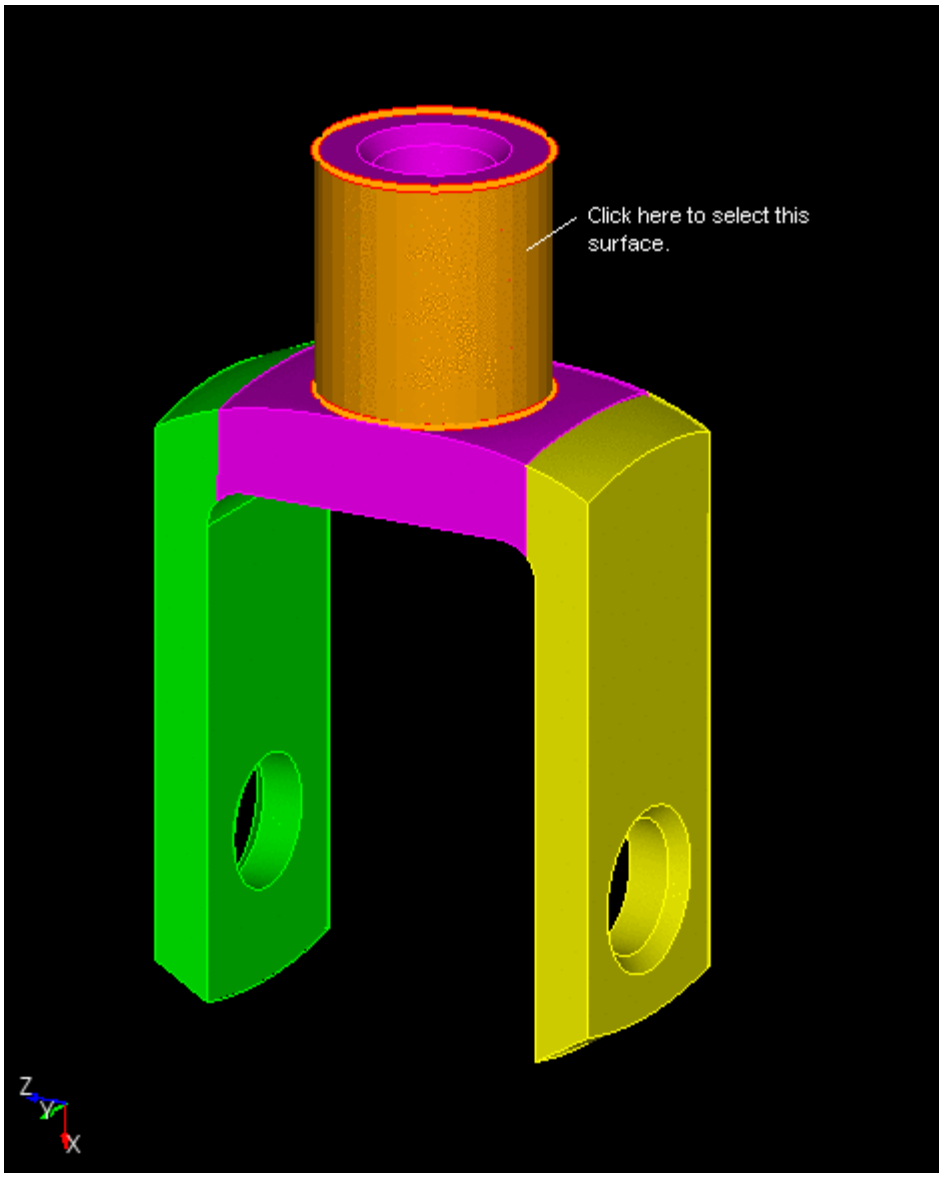
- Press the **Tweak Button** (since you are not tweaking Surface 46 directly, the surface does not need to be highlighted when you press the tweak button)

The **Geometry-Surface-Modify-Tweak** will open on the Control Panel as shown below.



- Enter **Surface 44** by typing it in at the input line or selecting from the graphics window
- Select the **Offset** option from the pulldown menu

Surface 44 is shown highlighted below.

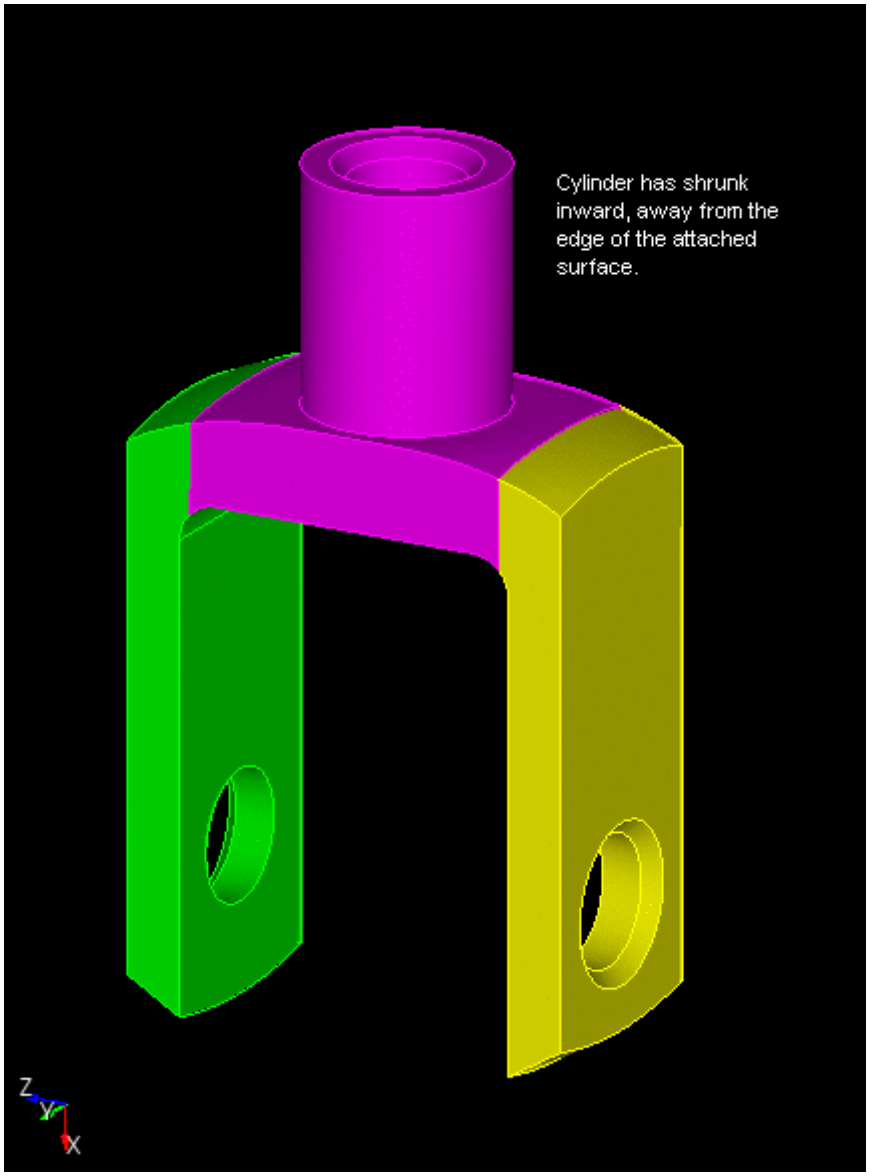


- Enter an Offset **Value** of **-0.9**.

The offset value is a percentage of the current size. Entering -0.9 will decrease the radius by 10 percent.

- Press **Apply**

The graphics window should now look like this. Notice that the radius of the cylinder has shrunk inward, increasing the gap between the edges on Surface 46.



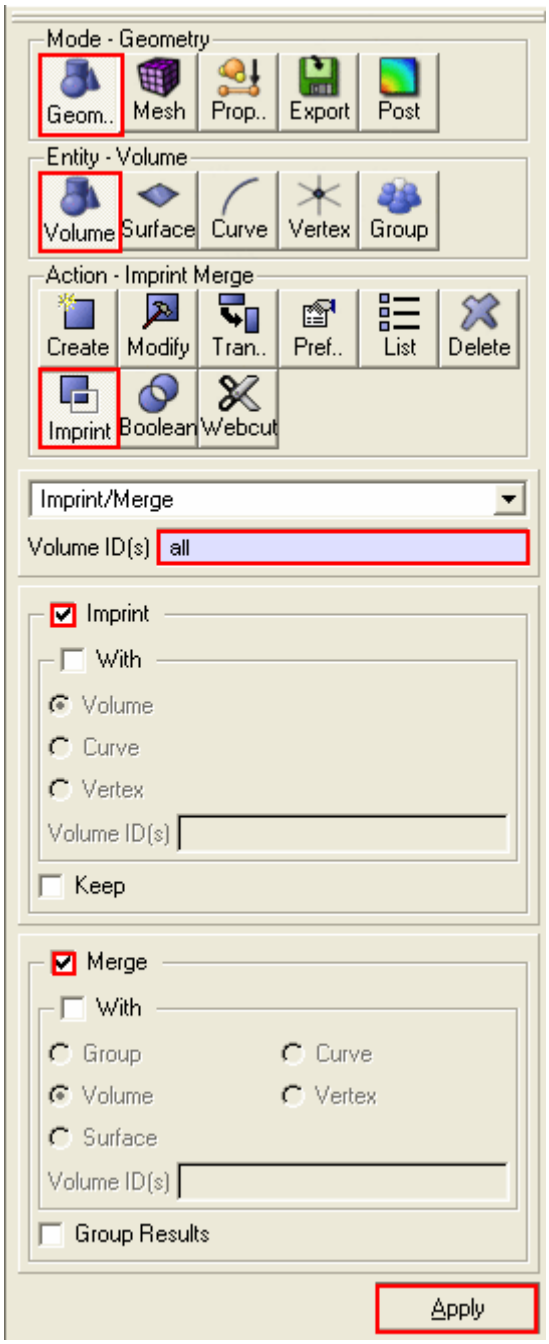
Power Tools GUI Tutorial

Step 9: Imprint/Merge

Imprinting is the process of projecting curves from one surface onto an overlapping surface. Merging is the process of taking two overlapping surfaces and merging them into one surface shared by two volumes, creating non-manifold geometry. Both imprinting and merging are necessary to make adjacent volumes have identical meshes at their intersection. Imprinting and merging is almost always necessary after webcutting.

- To open the imprint/merge menu, select the **Geometry** icon, then **Volume**, then **Imprint/Merge** on the Control Panel
- Enter **all** in the input window Check the **Imprint** and **Merge** boxes

- Press **Apply**



You will not notice any visible changes in the graphics window after imprint/merge operations, but results of the operations will be printed in the output window. Confirm that both surfaces have been merged by reading the output in the graphics window (You may have to scroll to see all of the results)

You can return to the Power Tools menu to see that the Close Loops and Overlapping Surfaces are gone.

- Press **Analyze** in the Power Tools menu

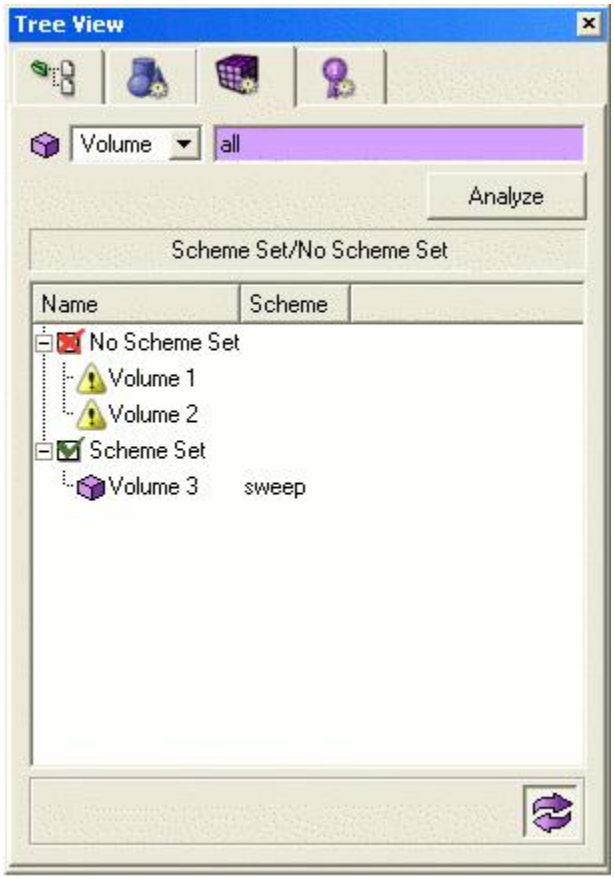
The display window will now read "Nothing Found" to indicate that are no geometry tests that fail.

Power Tools GUI Tutorial

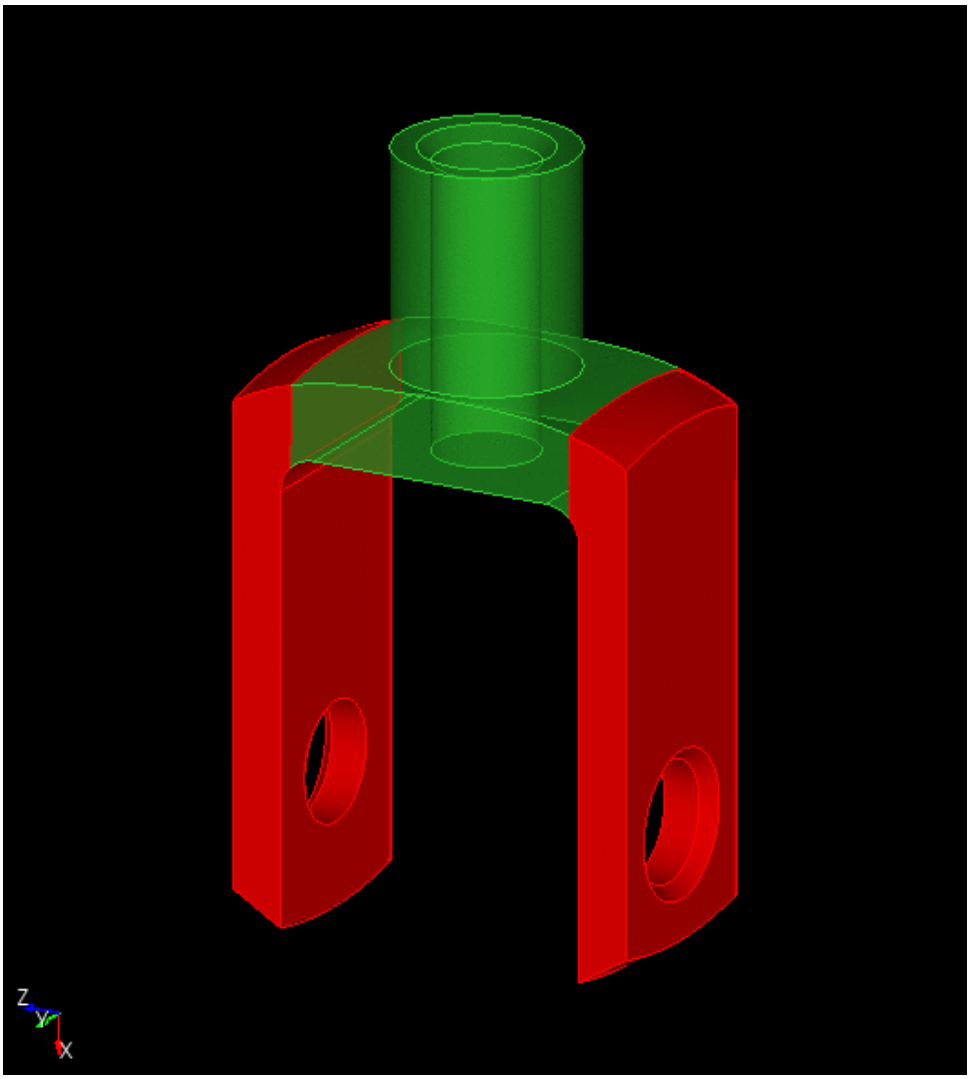
Step 10: Compositing Surfaces

Composite surfaces are adjacent surfaces that have been merged into one surface. Composite surfaces are created using Virtual Geometry, which is a built-in geometry kernel that sits on top of the existing geometry, and does not change the underlying geometry definition. Virtual geometry has the added advantage of being reversible. It can be removed after meshing. The general purpose for using composite surfaces is to deconstrain the mesh. For example, compositing two surfaces will remove the requirement that nodes be placed on the curve between the surfaces. Composite surfaces will be used in this example to facilitate the sweeping algorithm.

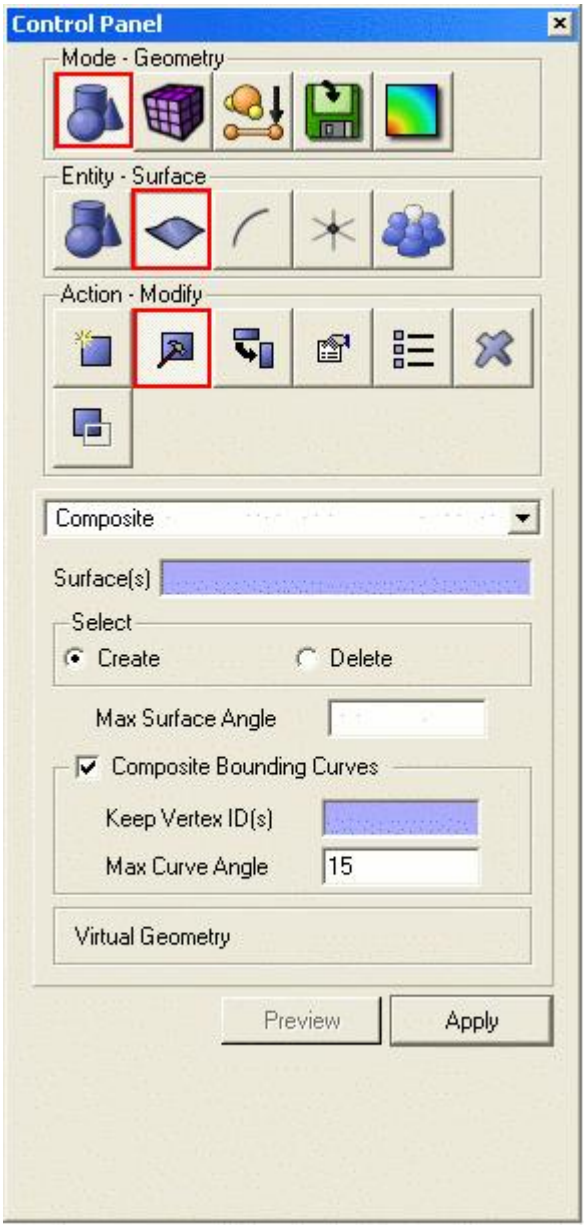
- Open the **Mesh Tools** tab
- Press **Analyze**
- Toggle the **Reset Graphics** button to show entities in green and red (for meshable and non-meshable volumes)



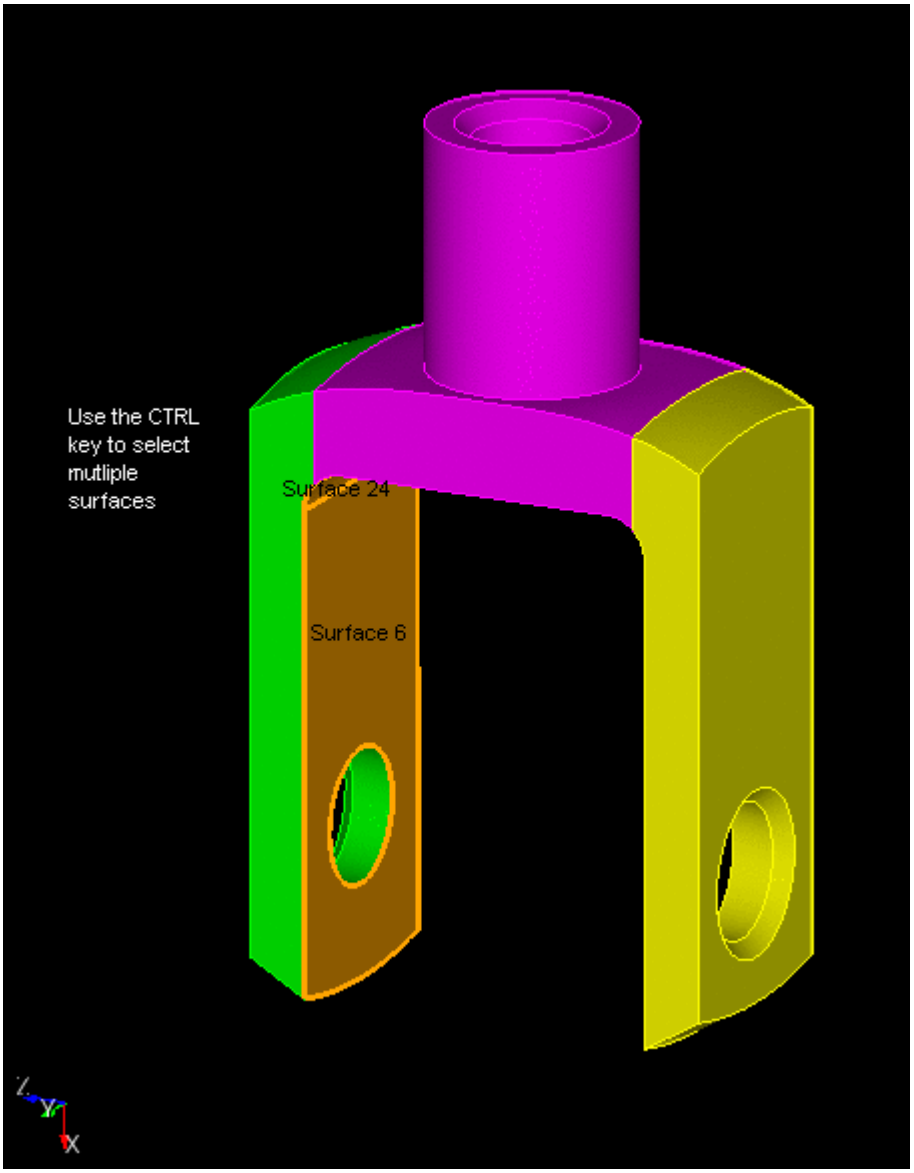
Two of the volumes have not had their scheme set. The graphics window should look like the following image. Red indicates that the volume scheme has not been set. Green indicates that the scheme has been set.



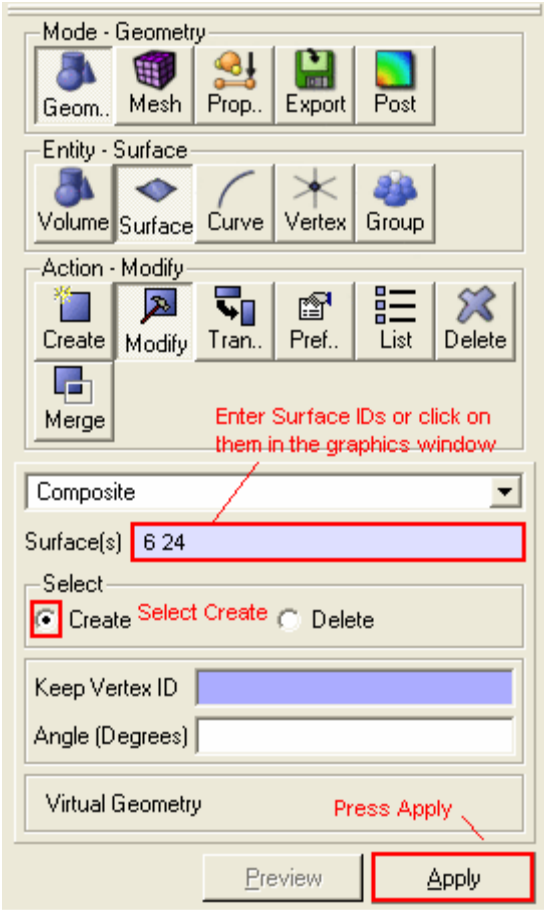
- Toggle the **Reset Graphics** button so it returns to the normal colors
- Open the **Geometry Tools** tab
- Press the **Composite Button** on the toolbar



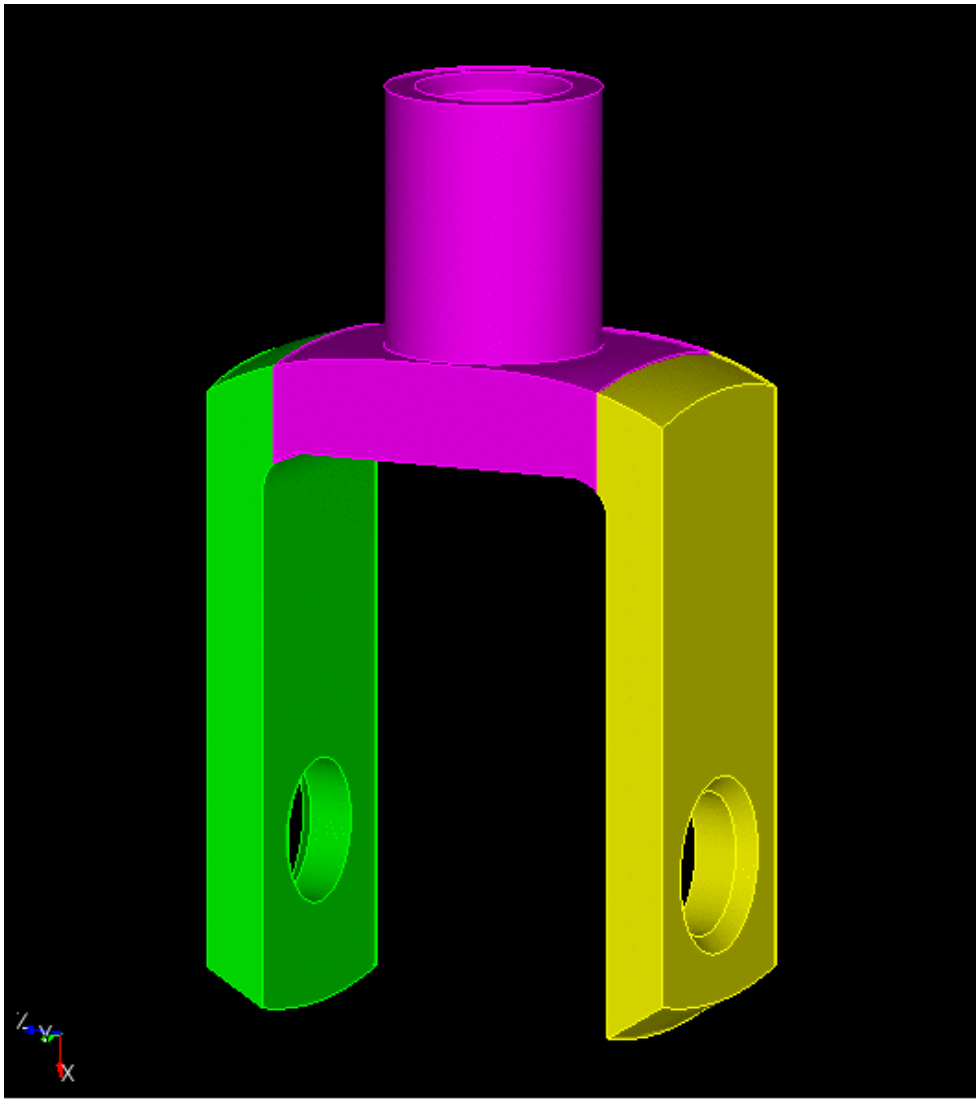
The Geometry-Surface-Modify-Composite menu will open on the Control Panel.



- Select **Surfaces 6** and **24** (shown in the image above) by entering them in at the input line, or by using CTRL-click in the graphics window
- Make sure the **Create** button is checked
- Press **Apply**

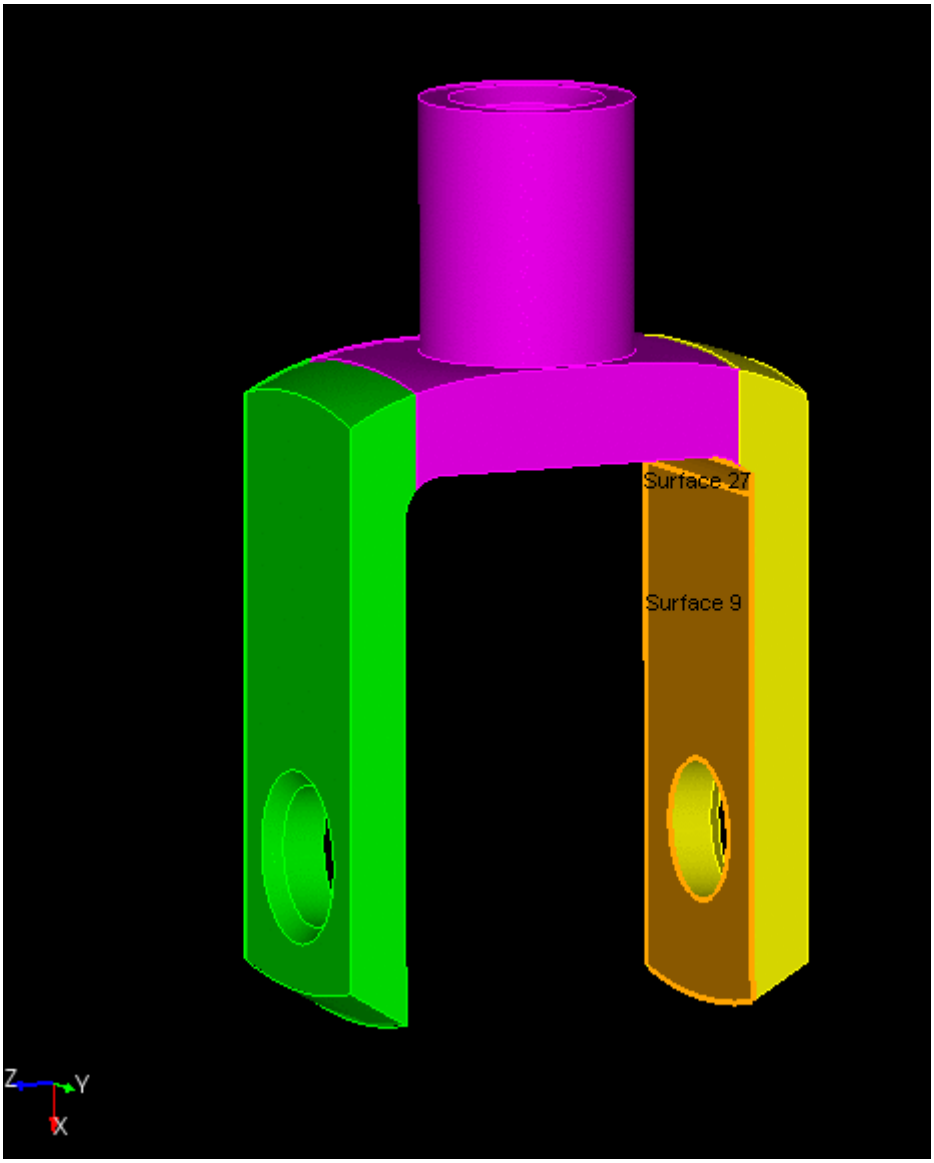


The two surfaces should appear merged, and the curve separating the surfaces should have disappeared.

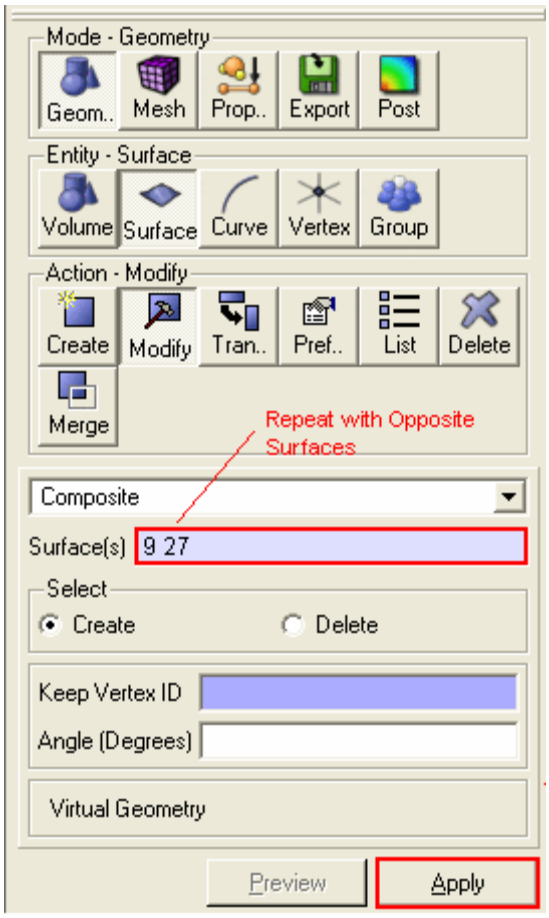


Repeat these steps with the opposite side.

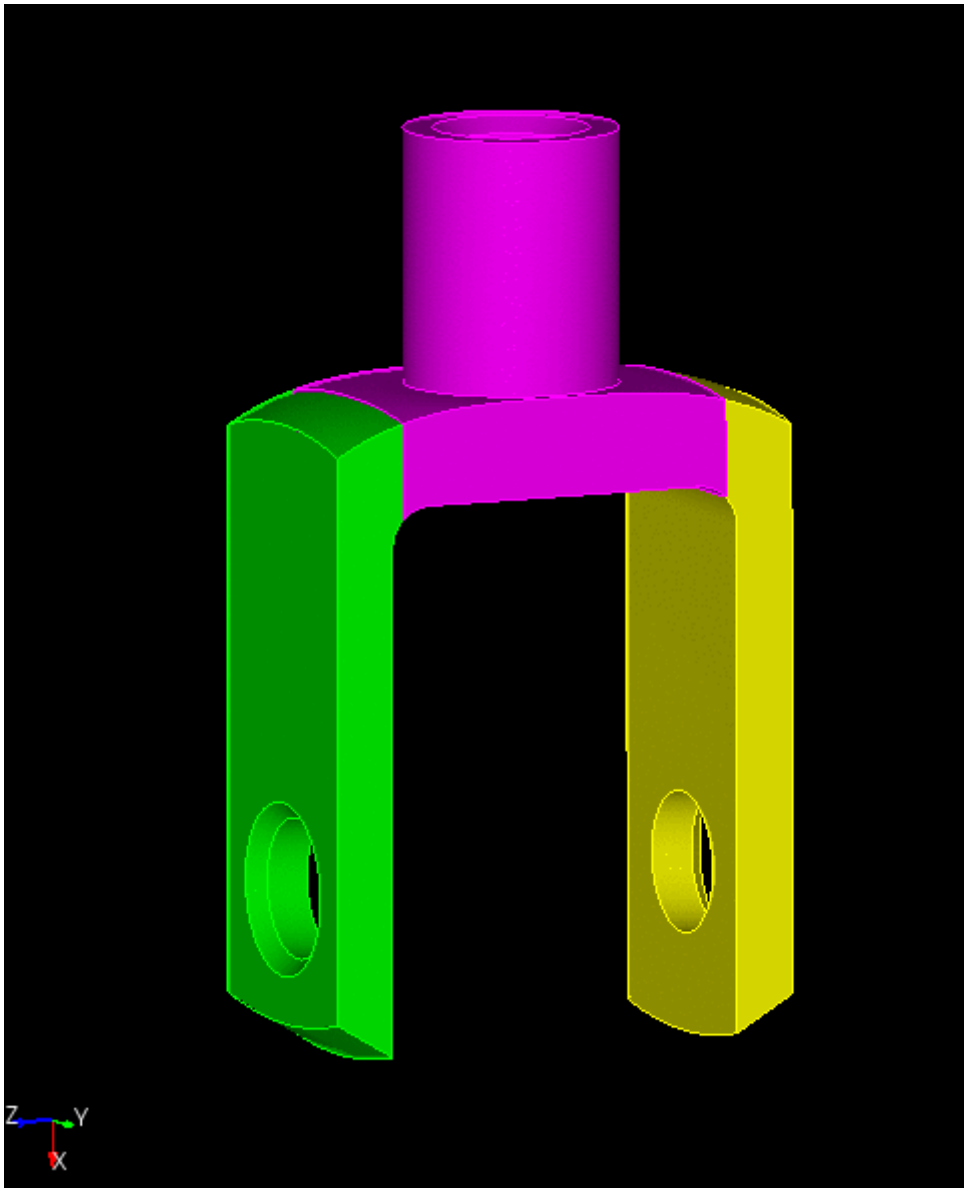
- Rotate the view window so **Surface 9** and **27** are visible



- Select **Surface 9** and **Surface 27** by using CTRL-click in the graphics window or entering in the input window
- Press Apply



Check to see that the surfaces have been composited and that your graphics window looks like the following image.

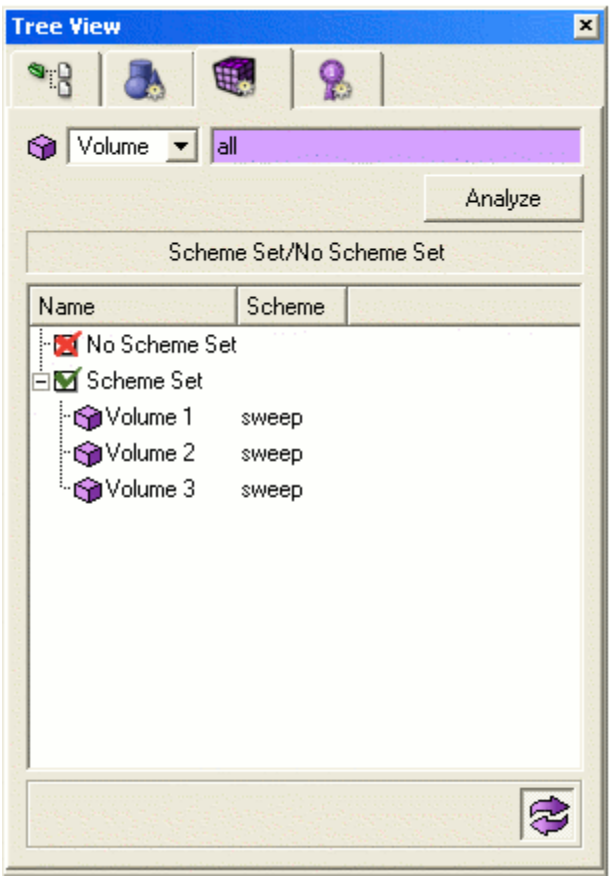


Power Tools GUI Tutorial

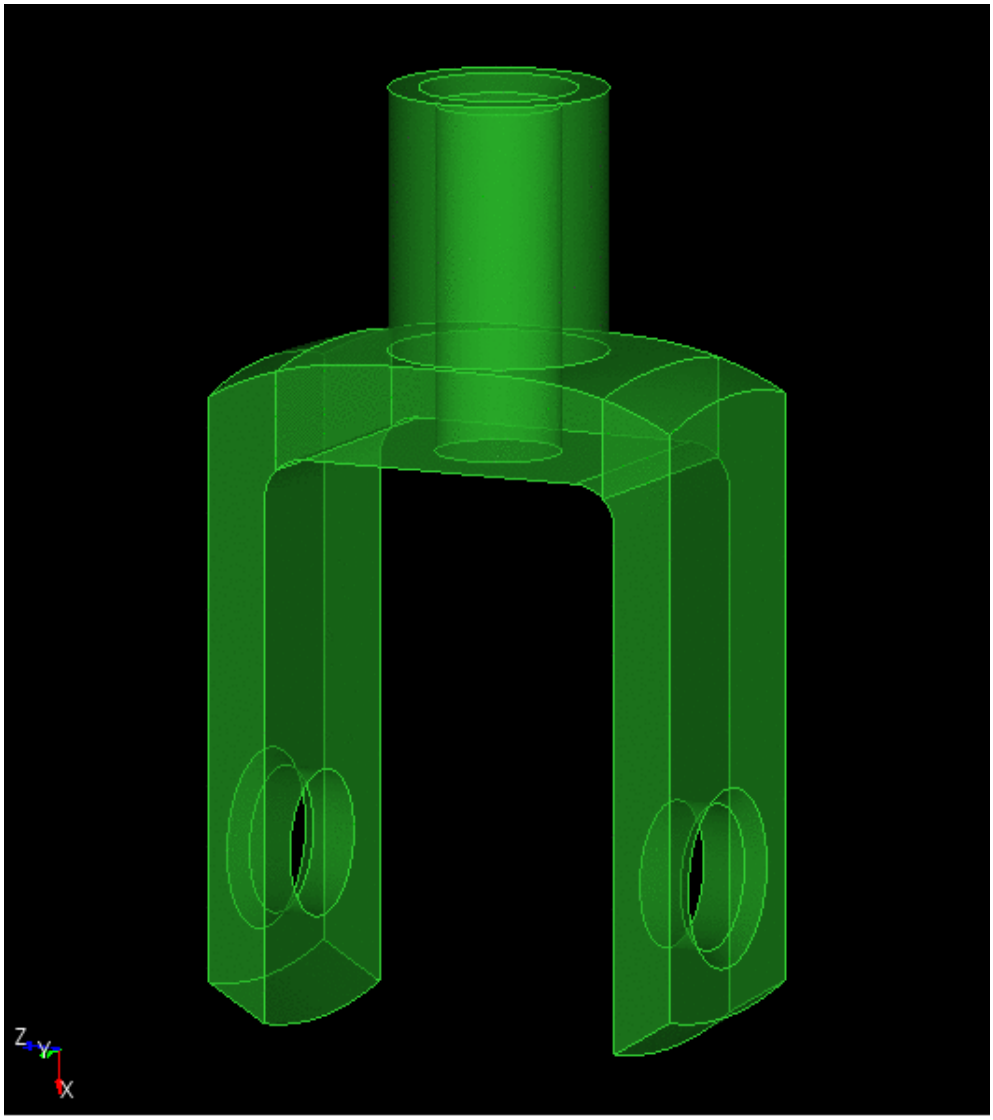
Step 11: Meshing the Model

Use the Mesh Power Tools to apply schemes to the remaining volumes.

- Press the **Mesh Tools** tab in the Power Tools window
- Press **Analyze**
- Toggle the **Reset Graphics** button



All of the schemes have now been set with a sweeping algorithm. The model is ready to be meshed. All volumes should appear green in the graphics window.

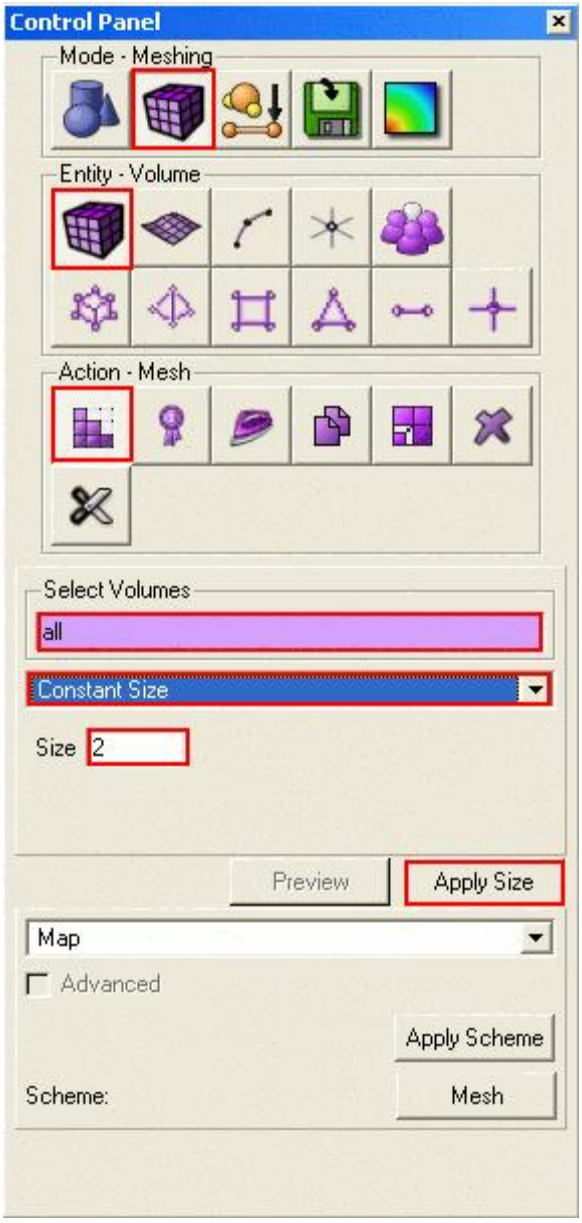


- Toggle the **Reset Graphics** button to return volumes to their original colors

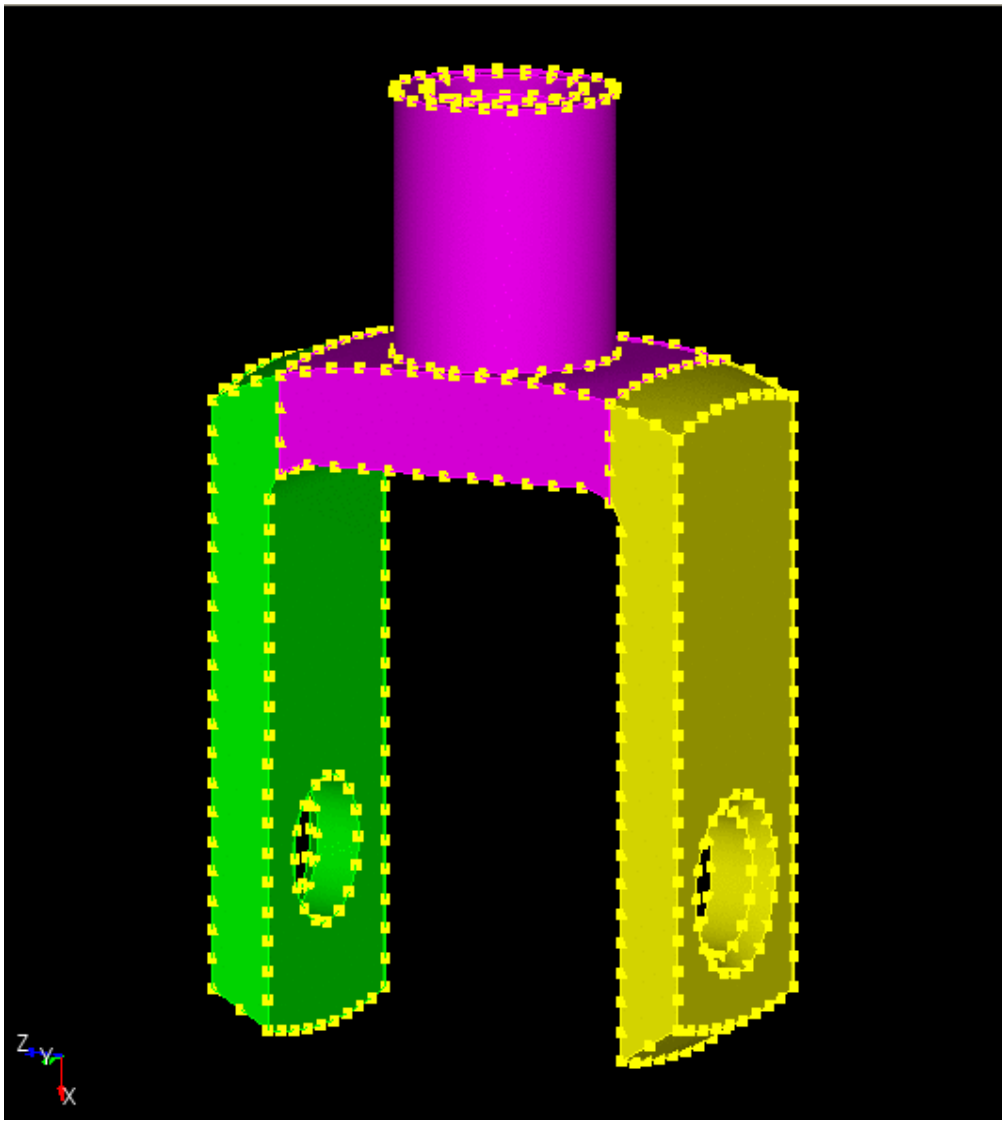
The meshing tools are located on the Control Panel under the purple Mesh Button

. Select **Volume** as the entity, and **Mesh** as the Action.

- Enter **all** in the "Select Volumes" input window
- Select **Constant Size** from the list of sizing options
- Enter **2** for the size
- Press **Apply Size**
- Press **Preview**

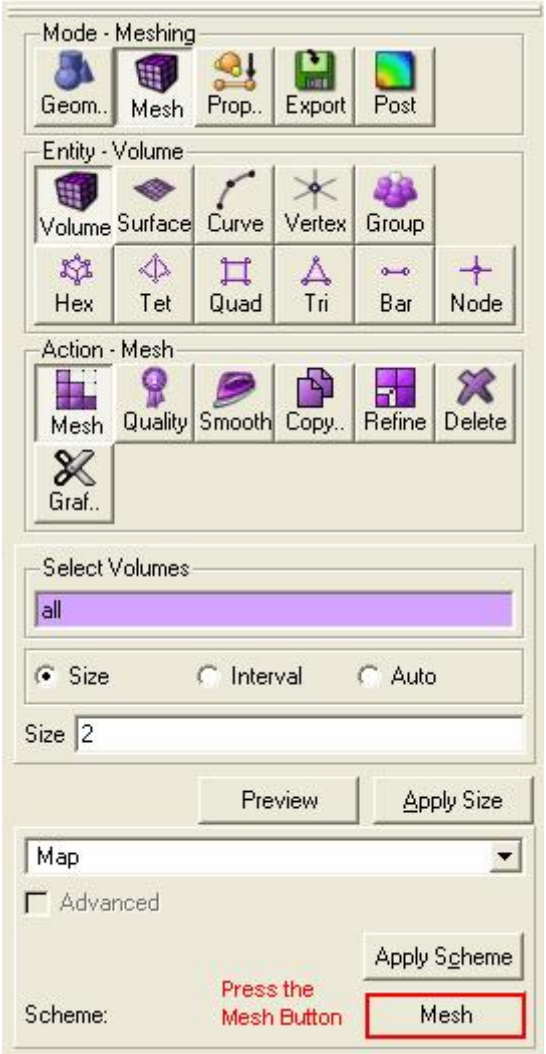


The graphics window should appear as follows, with the mesh size increments highlighted on all of the curves in the model.

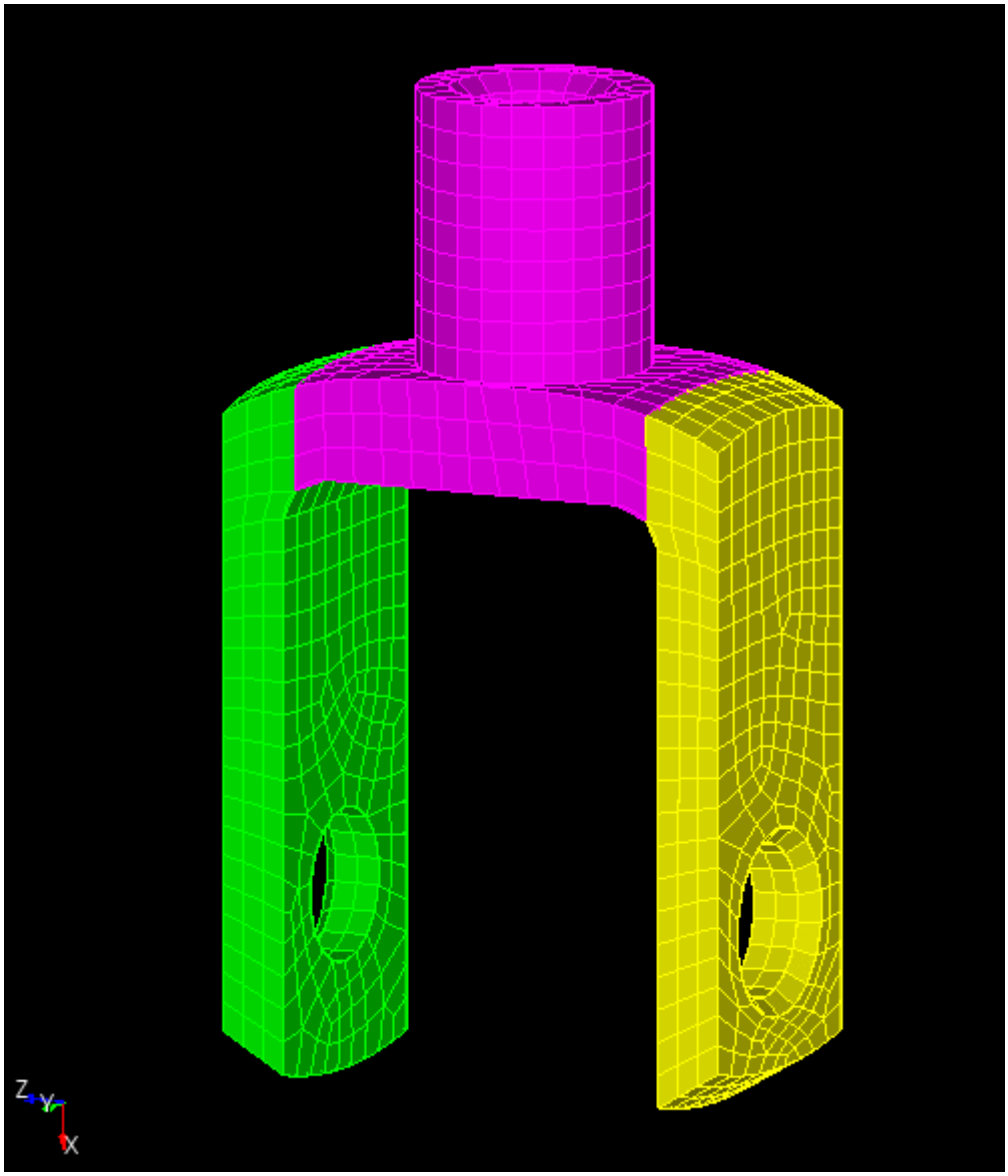


- Press the **Mesh** Button

There is no need to press the Apply Scheme button since the scheme have already been set in the Meshing Tools.



The final mesh should look like this:



Congratulations! You have just completed the Power Tools Tutorial. Click on the arrow to return to the Tutorial home.

Appendix

- Examples
- Available Colors
- Element Numbering
- Mesh Editing
- FullHex vs. NodeHex Representation
- APREPRO
- Periodic Space-filling Models (Tile)
- Troubleshooting
- References

Examples

- General Comments
- Simple Internal Geometry Generation
- Octant of Sphere
- Box Beam
- Thunderbird 3D Shell
- Advanced Tutorial

The purpose of this Appendix is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a few examples on the use of CUBIT. Some examples also demonstrate the use of the ACIS test harness as well as other related programs. This Appendix is not intended to be a step-by-step tutorial.

General Comments

The examples in this appendix show the use of CUBIT under various scenarios. To reproduce these examples, the user would need the journal files containing the CUBIT commands described below, and in some cases an ACIS SAT file containing model geometry. The journal files and SAT files necessary for running these examples are available from the CUBIT web site. For examples not requiring SAT files, the user can also type in the commands described for that example.

The examples in this appendix each cover several of CUBIT's mesh generation capabilities. The CUBIT features exercised by each example are shown in the table below:

Example	Geometry Features	Surface Meshing Features	Volume Meshing Features
Internal Geometry	Primitives, Booleans	Interval Assignment, Paving	Sweeping, Submapping
Sphere Octant	Primitives, Booleans, Decomposition, Merging	Interval Assignment, Paving, Triangle Tool, Smoothing	Sweeping, Submapping, Tetrahedron
Box Beam	Primitives, Merging	Interval Assignment, Mapping	
Thunderbird	Primitives, Booleans	Interval Assignment, Paving	
Advanced	Decomposition, Merging	Interval Assignment, Mapping,	Sweeping, Submapping,

Tutorial		Paving, Smoothing	Mapping

Example:Simple Internal Geometry Generation

This simple example demonstrates the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. It illustrates the brick, cylinder, subtract, pave and translate commands and boolean operations. The geometry to be generated is shown in Figure 1. This figure also shows the curve and surface labels specified in the CUBIT journal file. The final meshed body is shown in Figure 2. The CUBIT journal file follows. Note that the test of the journal file can be directly copied and pasted into the CUBIT command line.

```
# Internal Geometry Generation Example
Brick Width 10. Depth 10. Height 10. # Create Cube
Cylinder Height 12. Radius 3. # Create cylinder through Cube
View From 15 20 25 # Move to new Viewpoint
Display # You may want to move to graphics window to mouse
    # around to get the feel for it
Subtract 2 From 1 # Remove cylinder from cube--create hole
Body 1 Size 1.0 # Default element size for model
Label Curve On
Label Surface On # Turn on curve and surface labels for
            # scheme and size specification
Display
Surface 10 Interval 10 # Change intervals on cylinder surface
Curve 15 to 16 Interval 20 # Change intervals around
                # cylinder circle
Surface 11 Scheme Pave # Front surface paved
Volume 1 Scheme Sweep Source 11 Target 12
    # Remainder of block will be meshed by
    # sweeping front surface to back surface
Mesh Volume 1 # Create the mesh
Graphics Mode Hiddenline # Hiddenline view of cube (Figure 2)
```

The first two lines create a 10 unit cube centered at the origin and a cylinder with radius 3 units and height of 12 units also centered at the origin. The cylinder height is arbitrary as long as it is greater than the height of the brick. The subtract command then performs the boolean by subtracting the cylinder (body 2) from the block (body 1) to create the final geometry (body 1). The remainder of the commands simply assign the desired number of intervals and then generate the mesh. Note that since the cylindrical hole is a "periodic surface," there are no edges joining the two curves so the number of intervals along its axis must be set by the surface interval command.

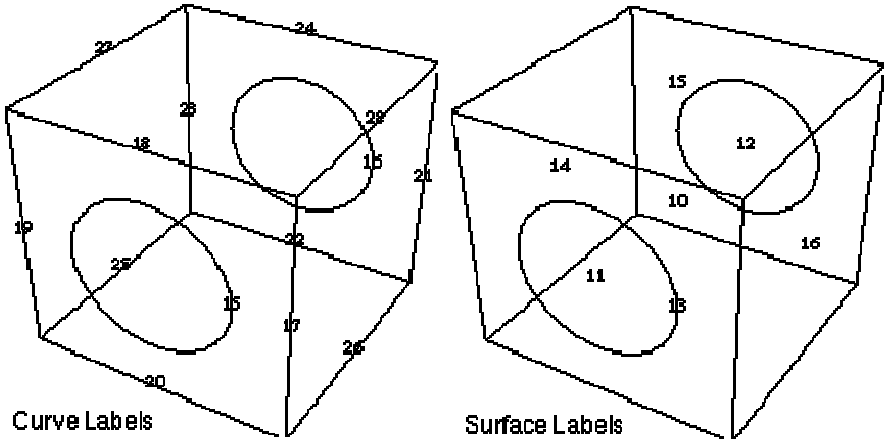


Figure 1. Geometry showing curve and surface entity Ids.

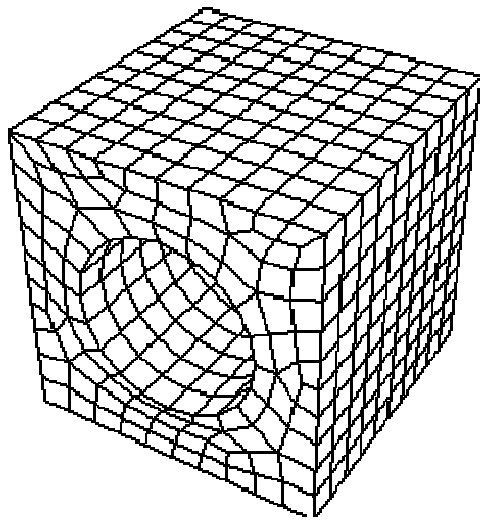


Figure 2. Geometry meshed with paving and sweeping schemes.

Meshing with Autoscheme

The above example is a useful example of manually setting intervals and meshing schemes. CUBIT's autoscheme feature simplifies the necessity to pick individual curves and surfaces and set individual meshing schemes. The following set of commands demonstrates the use of autoscheme to generate the same mesh. Note that not all geometries can successfully use the autoscheme feature:

```
# Internal Geometry Generation Example with Autoscheme
Brick Width 10. Depth 10. Height 10. # Create Cube
Cylinder Height 12. Radius 3. # Create cylinder through Cube
Subtract 2 From 1 # Remove cylinder from cube--create hole
Body 1 Size 1.0 # Default element size for model
Display
Volume 1 Scheme Auto # Let CUBIT choose the scheme
Mesh Volume 1 # Create the mesh
Graphics Mode Hiddenline # Hiddenline view of cube (Figure 2)
```

Example: Octant of a Sphere

This example also illustrates the internal geometry generation capabilities of CUBIT to generate an octant of a sphere. The procedure used is to generate the octant by creating a sphere only on the positive quadrant of the reference frame. Two methods of meshing are demonstrated in this example: one is to decompose the octant into two volumes - a central "core" and an outer "peel" which are both meshable using the sweep schemes. The second is to mesh the octant with the triprimitive and tetprimitive meshing schemes. This example uses the sphere, webcut, merge, auto, triprimitive, tetprimitive and smooth commands.

The following annotated CUBIT journal file will generate the meshes shown in Figure 1

Create an octant of a sphere on the positive quadrant
Sphere Radius 10.0 xpos ypos zpos #create the octant
Webcut Body 1 Cylinder Radius 4 Axis z Noimprint Nomerge
Coalesce redundant surfaces
Merge All
Volume 3 Size 0.4999
Volume 1 Size 0.6
Volume all Scheme Auto # Use auto to set meshing schemes
List Volume 1 #List the volume to see the schemes
List Volume 3 #List the volume to see the schemes
Mesh Volume all
Now try it with the tetrahedron this way
Reset
Sphere Radius 10. xpos ypos zpos # Create an octant
The tetprimitive scheme will mesh a tetrahedron with hexes
Volume 1 Scheme Tetprimitive # Mesh the volume with scheme tetprimitive
Surface All Scheme Triprimitive # Surfaces must be scheme triprimitive
Volume 1 size 0.7 # Set an interval size
Mesh Surface all # First mesh the surfaces
Smooth Surface all # Scheme Triprimitive often requires smoothing
Mesh Volume 1 # Mesh the volume
Export Genesis'Octant.gen' # Write out the mesh

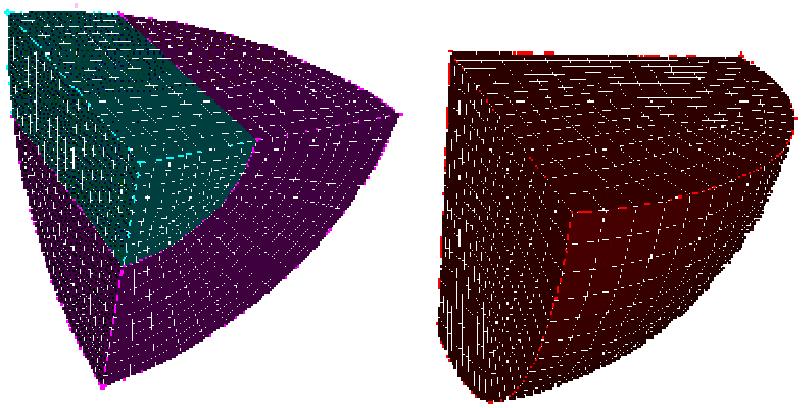


Figure 1. Octant of a Sphere Example Output

Example: Box Beam

A simple example using CUBIT is the box beam buckling problem shown in Figure 1. A description of an analysis which uses this type of mesh is found in (Lovejoy, 90). This example uses the merge, nodeset and block commands and the mapping mesh generation scheme. The geometry is generated inside of CUBIT using APREPRO commands and variables. The geometry file is as follows:

File: boxBeamGeom.jou
Side = {Side = 1.75}
Height = {Height = 12.0}
Upper = {Upper = 2.0}
Brick Width {Side/2.0} depth {Side/2.0} height {Height-Upper}
Body 1 name "lowerSection"
Brick Width {Side/2.0} depth {Side/2.0} height {Upper}
Body 2 name "upperSection"
Move lowerSection xyz {Side/4.0} {Side/4.0} {(Height-Upper)/2.0}
Move upperSection xyz {Side/4.0} {Side/4.0} {Upper/2.0 + Height - Upper}
Export acis "boxBeam.sat" #Save the file to SAT

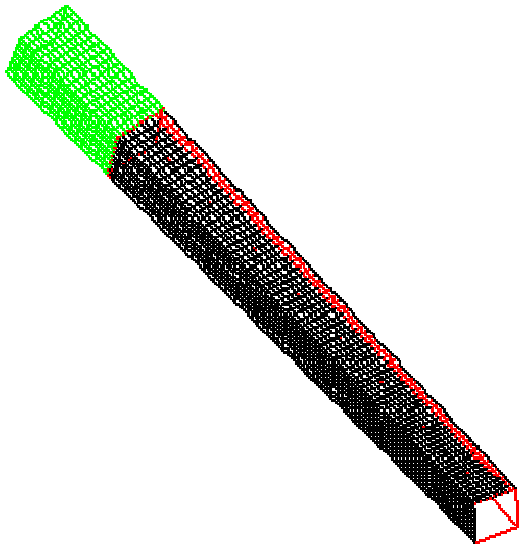


Figure 1. Box beam buckling example

In this example, it is assumed that subsequent analyses will take advantage of the problem symmetry and therefore only one-quarter of the box beam will be meshed. It is worth noting that there are a variety of ways to construct a solid model for this problem; however, experience thus far with ACIS and CUBIT indicates that the easiest way to model the box beam is to use ACIS block primitives. (Note that this geometry can also be generated using the internal CUBIT Brick primitive) Even though subsequent meshing will only be performed on the faces of the solid model, the entire 3D body is saved as an ACIS.sat file. The CUBIT journal file for the box beam example is:

```
# File: boxBeam.jou
# Thickness = {Thickness = 0.06}
# Crease = {Crease = 0.01}
# XYInts = {XYInts = 10}
# ZInts = {ZInts = 90}
# UpperInts = {UpperInts = 15}
Import Acis 'boxBeam.sat'
Merge All
Label Surface on
Label Curve on
Display
Curve 1 To 8 Interval {XYInts}
Curve 13 To 16 Interval {XYInts}
Curve 9 To 12 Interval {ZInts-UpperInts}
Curve 21 To 24 Interval {UpperInts}
Mesh Surface 3
Mesh Surface 6
Mesh Surface 9
Mesh Surface 12
NodeSet 1 Curve 1
NodeSet 2 Curve 4
NodeSet 1 Move {-Crease} 0 0
NodeSet 2 Move 0 {Crease} 0
Block 2 Surface 3
Block 2 Surface 6
Block 1 Surface 9
Block 1 Surface 12
Block 1 To 2 Attribute {Thickness}
Export Genesis 'boxBeam.exoII'
Quit
```

Commands worth noting in the CUBIT journal file include:

Block, Block Attribute

Allows the user to specify that shell elements for the surfaces of the solid model are to be written to the output (EXODUSII) database, and that shell elements be given a thickness attribute. This is necessary since CUBIT defaults to three-dimensional hexahedral meshing of solid model volumes.

NodeSet Move

Allows the user to actually move the specified nodes by a vector (Δx, Δy, Δz). This is advantageous for the buckling problem, since the numerical simulation requires a small "crease" in the beam in order to perform well.

Merge

Allows the user to combine geometric features (e.g. edges and surfaces).

Other commands in the journal file should be straightforward. Since the problem is sufficiently simple to mesh using a mapping transformation, specification of a meshing "scheme" is unnecessary (mapping is the default in CUBIT).

Finally, note that both the CUBIT journal files (boxBeamGeom.jou and boxBeam.jou) contain macros that are evaluated using APREPRO. The makefile is used to semi-automatically generate the mesh is given below. While this particular example is a trivial use of the software, it does serve to demonstrate a few of the capabilities offered by CUBIT.

```
# File: Makefile
boxBeam.g: boxBeam.exoII
ex2exlv2 boxBeam.exoII boxBeam.g
boxBeam.exoII: boxBeam.sat boxBeam.jou
cubit -batch -nographics boxBeam.jou
boxBeam.sat: boxBeamGeom.jou
cubit -batch -nographics boxBeamGeom.jou
boxBeam.jou: boxBeam.jou
clean:
@-rm *.sat *.exoII *.g
```

Example: Thunderbird

This example is the three-dimensional paving of a shell shown in Figure1. The 2D wireframe geometry of the thunderbird is given by the following FASTQ file:

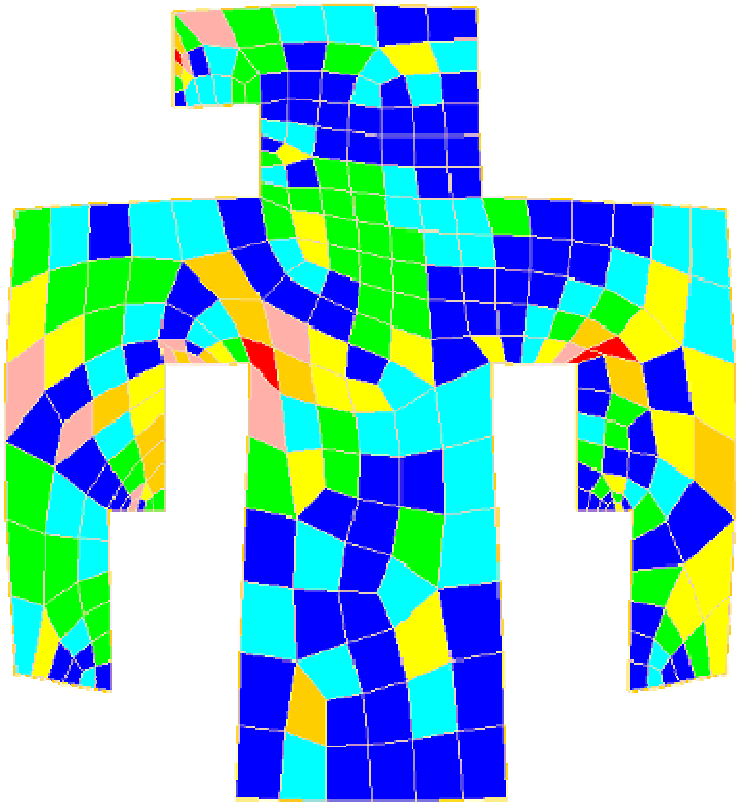


Figure 1. Sandia Thunderbird 3D shell

```
#File: tbird.fsq
TITLE
MESH OF SANDIA THUNDERBIRD

$ block {e = .2} int= {isq = 20}
$ number of elements in block thick {iblkt = 5 } block thickness {blkt=.2 }
$ block angle {angle=15}
$ magnification factor = {magnificationFactor=1.0}
$ bird {bthick = .018} {ithick = 3} {idepth = 20}
$ {pi = 3.14159265359} {rad=magnificationFactor/pi} {bdepth=1.}
$ preferred normalized element size = {elementSize=0.06}
$ number of intervals along outside edges =
$ {border_int=5} {corner_int=10} {side_int=20}
$ {outsideIntervals= 2*corner_int+side_int}
$ {boxTop=.2} {topIntervals = 8}

$ {insideCurveInt=8}

$ {MAG=magnificationFactor/3.0}

$ {middleInside=MAG*0.97}
$ {xCurveStartInside=MAG*0.60}
$ {yCurveStartInside=MAG*0.93}
$ {curveMiddleInside=MAG*0.81}

$ {xCurveStartOutside=MAG*0.75}
$ {yCurveStartOutside=MAG*1.17}
$ {middleOutside=MAG*1.20}
$ {curveMiddleOutside=MAG*1.01}
$ {boundingBox = MAG*1.5}

$ Thunderbird Coordinates
```

```
POINT 1 {MAG*-.40} {MAG*.78}
POINT 2 {MAG*-.40} {MAG*.59}
POINT 3 {MAG*-.22} {MAG*.59}
POINT 4 {MAG*-.22} {MAG*.40}
POINT 5 {MAG*-.75} {MAG*.40}
POINT 6 {MAG*-.78} {MAG*-.09}
POINT 7 {MAG*-.75} {MAG*-.58}
POINT 8 {MAG*-.53} {MAG*-.60}
POINT 9 {MAG*-.54} {MAG*-.23}
POINT 10 {MAG*-.42} {MAG*-.23}
POINT 11 {MAG*-.42} {MAG*.07}
POINT 12 {MAG*-.24} {MAG*.07}
POINT 13 {MAG*-.27} {MAG*-.80}
POINT 14 {MAG*.27} {MAG*-.80}
POINT 15 {MAG*.24} {MAG*.07}
POINT 16 {MAG*.42} {MAG*.07}
POINT 17 {MAG*.42} {MAG*-.23}
POINT 18 {MAG*.54} {MAG*-.23}
POINT 19 {MAG*.53} {MAG*-.60}
POINT 20 {MAG*.75} {MAG*-.58}
POINT 21 {MAG*.78} {MAG*-.09}
POINT 22 {MAG*.75} {MAG*.40}
POINT 23 {MAG*.22} {MAG*.40}
POINT 24 {MAG*.21} {MAG*.78}
POINT 25 {MAG*0.0} {MAG*.80}

$ lines for Tbird

LINE 1 STR 1 2
LINE 2 STR 2 3
LINE 3 STR 3 4
LINE 4 STR 4 5
LINE 5 CIRM 5 7 6
LINE 6 STR 7 8
LINE 7 STR 8 9
LINE 8 STR 9 10
LINE 9 STR 10 11
LINE 10 STR 11 12
LINE 11 STR 12 13
LINE 12 STR 13 14
LINE 13 STR 14 15
LINE 14 STR 15 16
LINE 15 STR 16 17
LINE 16 STR 17 18
LINE 17 STR 18 19
LINE 18 STR 19 20
LINE 19 CIRM 20 22 21
LINE 20 STR 22 23
LINE 21 STR 23 24
LINE 22 STR 24 1 0 7 1.0

$ REGIONS

SIZE {elementSize*MAG}

REGION 1 1 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 *
-16 -17 -18 -19 -20 -21 -22

SCHEME 0 X
BODY 1
EXIT
```

A command interpreter has been developed inside CUBIT to convert FASTQ geometry into CUBIT's modeling system (ACIS). The previous file, tbird.fsq, can be read into CUBIT by the command:

```
Import Fastq "<file_name>"
```

The file can be read into CUBIT and converted from a 2D "sheet" body to a 3D solid, by the following commands:

```
#File: tbird3dGeom.jou
import fastq "tbird.fsq"
cylinder radius.5 height 1.25
rotate body 2 about x angle 90
sweep surface 1 vector 0 0 1 distance 1
intersect body 3 with body 2
export acis "tbird3d.sat"
```

Example: Advanced Tutorial

The objective of this example is to illustrate the use of some advanced meshing operations to mesh a more complex geometry. The example purposely does not do everything right the first time to demonstrate the thought process a user would go through when meshing a real part for the first time. This example demonstrates the use of webcut to decompose the model into sweepable volumes, manually setting meshing schemes when scheme auto fails for certain volumes and matching intervals to ensure meshing scheme constraints are met. It should be noted that the sequence of commands is important to successfully generate the meshed model. It is recommended that the user first perform all the decomposition on the model, then imprint the entire model. Imprinting ensures that the topology of adjacent bodies match so that correct merging of adjacent surfaces can be performed. Next, use the merge all command to merge the common surfaces and ensure a contiguous mesh throughout the model. It is important to watch the merge all command output, since during typical merge all operations, all of the curves and vertices will be merged during the surface merging. Thus unless specifically desired, curve and vertex merging messages should not be seen from this command. If these are reported during the execution of the command, it may indicate invalid topology (remedied by an imprint all) or some other invalidity in the model. Performing an imprint all after the merge all may corrupt the data base; the user should not perform geometry operations after the merge command. Next, set the element size (e.g. volume all size 15) then the meshing schema (e.g. volume all scheme auto). The regime is finished when the mesh command is issued. Setting up BC's and Element Blocks are not covered in this tutorial.

The command 'set default names on' assigns names to the geometric entities. These names are saved with the geometry when the file is saved and also remain constant within code revisions. Throughout the session, each entity will acquire multiple names and any name given for each entity is valid for identification.

The ACIS SAT file for this tutorial can be obtained by contacting the cubit development team cubit-dev@sandia.gov

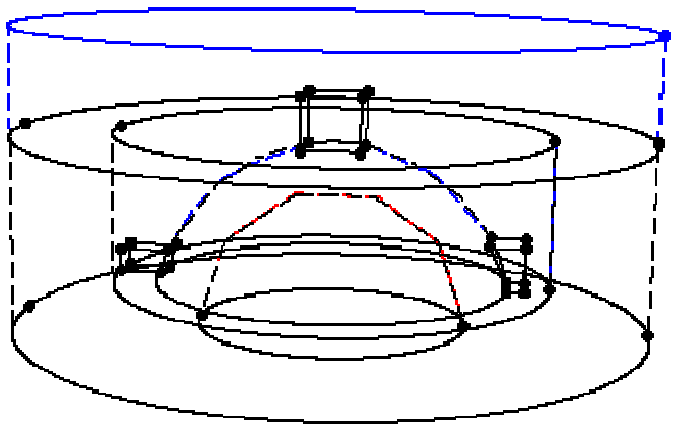


Figure 1. Geometry of Advanced Tutorial

The geometry used in this example is shown in Figure 1 above. The journal file for this exercise is given in the following file:

advanced_tutorial.jou

The resulting mesh is shown in Figure 2.

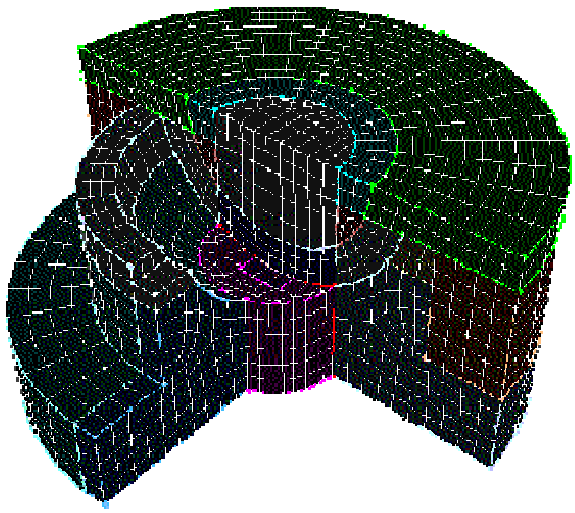


Figure 2. Mesh of Advanced Tutorial Problem

Alpha Commands

CUBIT has several functions that are currently in development and are considered "Alpha" features. These are as follows

- Automatic Detail Suppression
- Automatic Geometry Decomposition
- Feature Size
- Hex Tet
- Hex to Void
- Optimize Jacobian
- Plastering
- Randomize Smoothing
- Sculpting
- Stride
- Super Sizing Function
- Test Sizing Function
- Transition
- Untangle Smoothing
- Whisker Weave

Automatic Detail Suppression

Geometry models often have small features, which can be difficult to resolve in a mesh. In fact, these features are sometimes too small to see, and are revealed only when the user attempts to mesh the geometry. Automatic detail suppression identifies and removes the following types of features from the geometric model:

- valence-2 vertices
- short edges
- small faces

Details are removed using virtual geometry , which means they can be restored later if desired.

There are several stages to the automatic detail suppression process, all of which can be controlled separately by the user. Small details are identified using the command:

Detail <ref entity list> [identify] [dimension <dim> [only]]

The results are placed in a series of groups named "detail_vertices", "detail_edges", "detail_faces" and "detail_volumes". These details can be drawn or highlighted using the normal group commands:

Draw {detail_vertices | detail_edges | detail_faces | detail_volumes}

Highlight {detail_vertices | detail_edges | detail_faces | detail_volumes}

Or by using the following command:

Detail <ref entity list> draw [dimension <dim> [only]]

Details are removed automatically from the model using the command:

Detail <ref entity list> remove [dimension <dim> [only]]

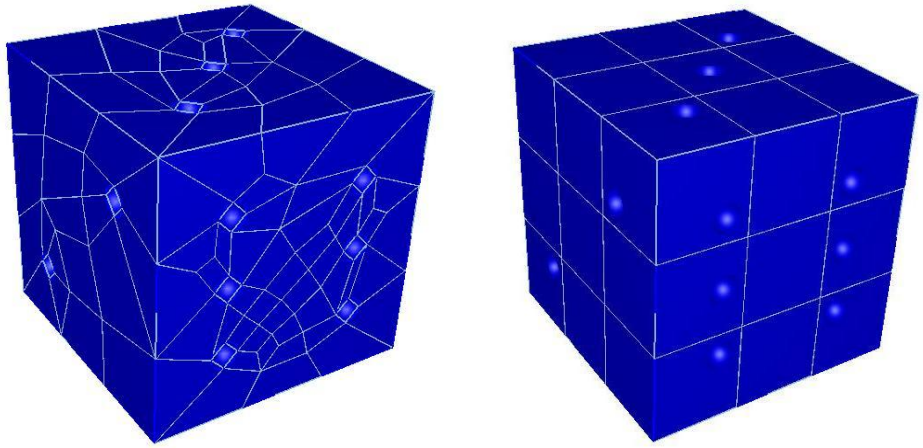
The **dimension** option is used to identify the maximum dimension of entities examined for small detail identification (<**dim**> is **3**, **2**, **1** for volumes, surface, and curves, respectively). If the **only** identifier is specified, only entities of the specified dimension are examined, otherwise that dimension and all lower dimensions are examined.

In some cases, details are identified which the user would like to retain in the model; likewise, the algorithm used to identify small details sometimes misses small details the user would like removed from the model. To include or exclude geometric entities from the list of small details to be removed, the following command is used:

Detail <ref entity list> [include | exclude]

Example

Shown below is a model of a game die meshed with identical mesh size, with details included (left) and removed (right).



Note: "Small" Measurement

Automatic detail suppression identifies "small" geometric entities by comparing their "size" to the mesh size assigned by the user to the entity. Anything smaller than that size is identified as being a detail and put in the appropriate detail group (e.g. detail_faces, detail_edges, etc.). The size of an edge is simply its arc length; surfaces and volumes are measured using the "hydraulic diameter" (see next note).

Note: Hydraulic Diameter

The hydraulic diameter of a surface is computed as 4.0*A/P, where A is the surface area and P is the summed arc lengths of all bounding curves. For circles, the hydraulic diameter is the circle diameter; for squares, it is the length of the bounding curves. Similarly, for volumes, the hydraulic diameter is computed as 6.0*V/A, which evaluates to the diameter and bounding curve length for perfect spheres and cubes, respectively.

Automatic Geometry Decomposition

In many cases, model geometry includes protrusions which, when cut off using geometry decomposition, are easily meshable with existing algorithms. CUBIT includes a feature-based decomposition capability, which automates this process. This algorithm operates by finding concave curves in the model, grouping them into closed loops, then forming cutting surfaces based on those loops. Although this algorithm is still in the research stage, it can be useful for automating some of the decomposition required for typical models.

To automatically decompose a model, use the command

Cut Body <body_id_range> [Trace {on|off}] [Depth <cut_depth>]

If the **Trace** option is used, the algorithm prints progress information as decomposition progresses. The **Depth** option controls how many cuts are made before the algorithm returns; by default, the algorithm cuts the model wherever it can.

Automatic decomposition is used to decompose the model shown in Figure 1 (left), with the results shown in Figure 1 (right). In this case, automatic decomposition performs all but one of the required cuts.

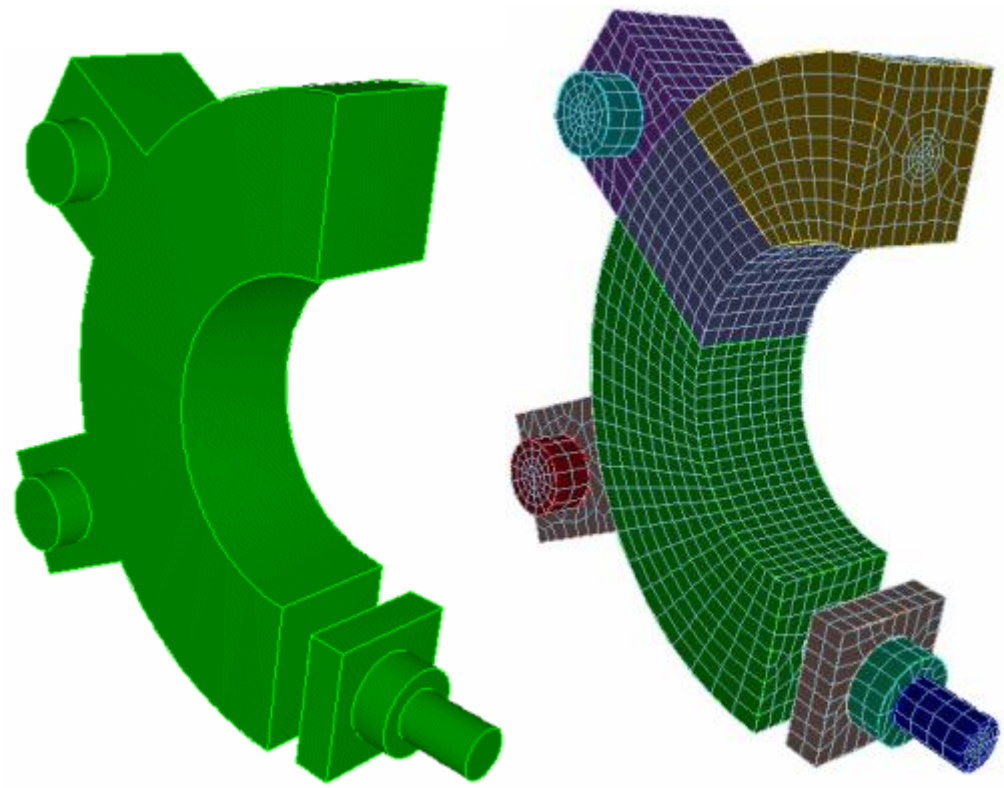


Figure 1. Model where automatic decomposition was utilized.

FeatureSize

Applies to: Curves

Summary: Meshes a curve based on its proximity to nearby geometry and size of nearby geometric features. This is an alpha feature and should be used with caution.

Syntax:

Curve <range> Scheme Featuresize

Related Commands:

Curve <range> Density <density_factor>

Discussion:

The user may also automatically bias the mesh from small elements near complicated geometry to large elements near expanses of simple geometry. Meshing a curve with scheme featuresize places nodes roughly proportional to the distance from the node to a piece of geometry that is foreign to the curve. Foreign means that the geometric entity doesn't contain the curve, or any of its vertices (ie. the entity's intersection with the curve is empty). It is known that featuresize is a continuous function that varies slowly. Featuresize meshing is very automatic and integrated with interval matching. Featursesize meshing works well with paving, and in some cases with structured surface-meshing schemes (map, submap) as well.

If desired, the user may specify the exact or goal number of intervals with a size or interval command, and then the featuresize function will be used to space the nodes.

The featuresize function may also be scaled by the user to produce a finer or coarser mesh using the **density** command as follows:

Curve <range> Density <density_factor>

The default scaling factor or **density** is 1. Higher densities also reduce the transition rate of the node spacing. A density of 2 usually gives a good quality mesh. A density below about 0.5 could produce rapid transitions and poor mesh quality. The following shows an example of different density values when using the featuresize scheme.

HexTet

Applies to: Volumes

Summary: Meshes a volume using the HexToVoid scheme until hex meshing terminates, finishes remaining void with scheme TetMesh. This is an alpha feature and should be used with caution.

Syntax:

Volume <range> Scheme Hextet

Related Commands:

Set Hextet Transition_type {two_triangle|four_triangle|pyramid }

Discussion:

This algorithm (Tuchinsky, 97; Meyers, 98) combines the HexToVoid and TetMesh schemes to create a complete, mixed-element mesh. The HexToVoid algorithm is first invoked to create as many hexes as possible in the volume, working inward from the boundaries. The quadrilateral boundary is then converted to a triangular boundary, and the remaining volume is filled with tetrahedra.

There are three methods currently supported to convert the quadrilateral void boundary to triangles. The two-triangle scheme subdivides each quadrilateral along its shortest diagonal, producing two boundary triangles. The four-triangle scheme inserts a center point on the quadrilateral and divides into four triangles. The pyramid scheme inserts a layer of pyramid elements on top of the quadrilaterals, resulting in a fully-conformal mesh.

HexToVoid

Applies to: Volumes

Summary: Meshes a volume building hexes from the exterior surfaces of the volume until hexes can no longer be inserted. In general this scheme will not produce a complete (closed) mesh for arbitrary volumes. This is an alpha feature and should be used with caution.

Syntax:

Volume <range> Scheme Hextovoid

Related Commands:

Mesh Volume <range> [Hexes <number_hexes>]

Mesh Volume <range> [Layers <number_layers>]

Discussion:

This algorithm is related to plastering, except that the exterior mesh remains fixed. In the general case, this will result in a partial all-hex mesh, and one or more unmeshed interior void regions. The algorithm currently successfully creates a partial mesh for most geometries. HexToVoid is most often used as part of the HexTet meshing algorithm discussed later.

A partial mesh can be created instead of meshing the complete geometry. The number of hex elements or the number of completed layers inward can be specified at the command line.

Optimize Jacobian

Applies to: Volume meshes

Summary: Produces locally-uniform hex meshes by optimizing element Jacobians

Syntax:

Volume <range> Smooth Scheme Optimize Jacobian [param]

Discussion:

The Optimize Jacobian method minimizes the sum of the squares of the Jacobians (i.e., volumes) attached to the smooth node. Meshes smoothed by this means tend to have locally-uniform hex volumes.

The parameter <**param**> has a default value of 1, meaning that the method will attempt to make local volumes equal. The parameter, which should always be between 1 and 2 (with 1.05 recommended), can be used to sacrifice local volume equality in favor of moving towards meshes with all-positive Jacobians.

Plastering

Applies to: Volumes

Summary: Research algorithm for generating all-hexahedral meshes for arbitrary 3D volumes. This is an alpha feature and should be used with caution.

Syntax:

Volume <range> Scheme Plaster

Related Commands:

Mesh Volume <range> [Hexes <number_hexes>] [Layers <number_layers>]

Discussion:

Plastering (Blacker, 93) uses the discretized surface and projects elements into the interior of the volume. This continues until the volume fills, with adjustments made to the exterior surface mesh as deemed necessary. This algorithm is currently under development and not suggested for use although it may be tested if desired. It should currently perform well for blocky structures where the surface mesh will form a valid boundary for an interior hex mesh. Some examples of these structures are shown in the following figure. These structures allow very straightforward hex element connectivity and do not contain any irregular nodes (nodes that are shared by other than four element edges in a given layer).

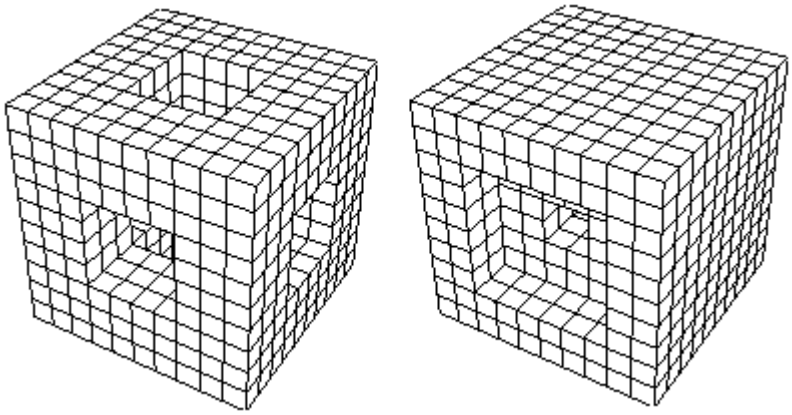


Figure 1. Plastering Examples

A partial mesh can be created instead of meshing the complete geometry. The number of hex elements or the number of completed layers inward can be specified at the command line when giving the mesh command by using the following syntax:

Mesh Volume <range> [Hexes <number_hexes>] [Layers]

Randomize

Applies to: Curve, Surface and Volume meshes

Summary: Randomizes the placement of nodes on a geometry entity

Syntax:

{Surface|Volume} <range> Smooth Scheme Randomize [percent]

Discussion:

This scheme will create non-smooth meshes. If a percent argument is given, this sets the amount by which nodes will be moved as a percentage of the local edge length. The default value for percent is 0.40. This smooth scheme is primarily a research scheme to help test other smooth schemes.

Sculpting

Applies to: Volumes

Summary: Grid based/Inside-Out research algorithm for generating all-hexahedral meshes for arbitrary 3D volumes. This is an alpha feature and should be used with caution.

Syntax:

Volume <range> Scheme Sculpt

Related Commands:

Sculpt <volume_id_range> from <volume_id_range>

Discussion:

Sculpting takes a grid based approach to creating a volumetric mesh by
surrounding the meshing geometry with a structured grid, removing elements that lie outside the volume boundary from the grid, manipulating the resulting stairstep mesh, and smoothing the exterior nodes to the volume boundary. The Sculpt command can be used when a user desires to define there own bounding grid to build the volume mesh from. Multiple volumes can define the user defined boundary grid. Currently Sculpting is still in stages of research and development.

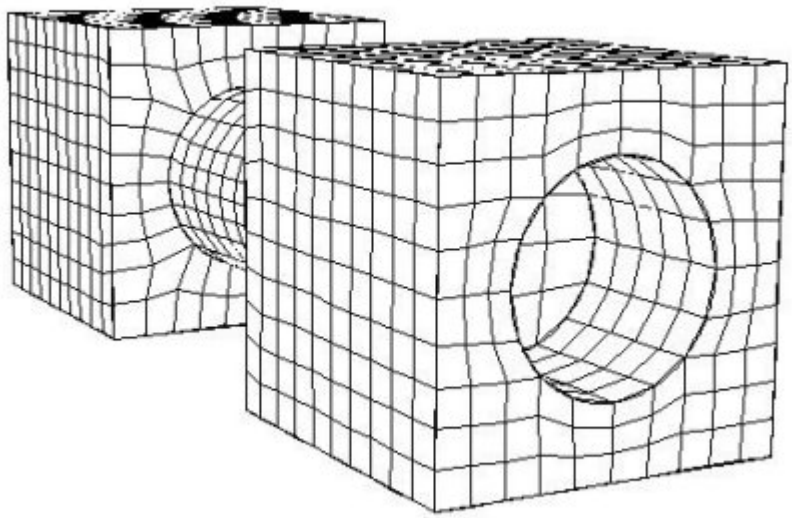


Figure 1. Sculpted mesh of a dumbbell shape

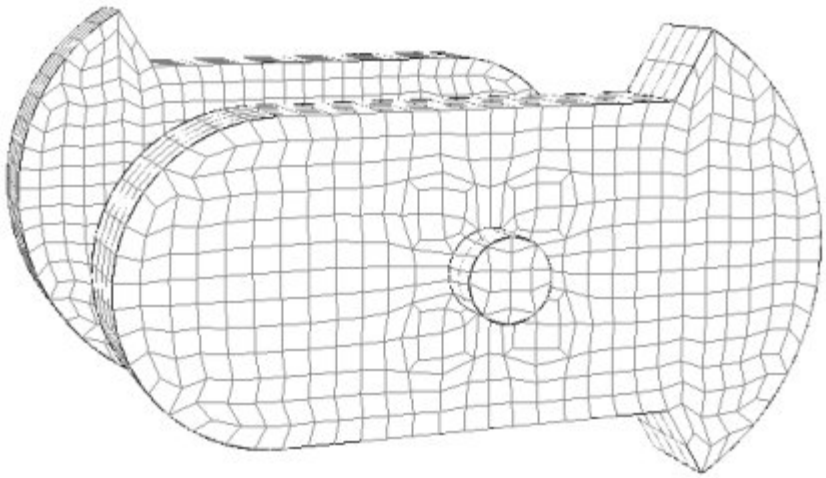


Figure 2. Sculpted mesh of a mechanical part

Stride

Applies to: Curves

Summary: Mesh a curve with node spacing based on a general field function. This is an alpha feature and should be used with caution.

Syntax:

Curve <range> Scheme Stride

Discussion:

The ability to specify the number and location of nodes based on a general field function is also available in CUBIT. With this capability the node locations along a curve can be determined by some field variable (e.g. an error measure). This provides a means of using CUBIT in adaptive analyses. To use this capability, a sizing function must have been read in and associated to the geometry (See Exodus II -based field function for more information on this process). After a sizing function is made available, the *stride* scheme can be used to mesh the curves.

Super Sizing Function

The **Super** sizing function computes both the Curvature and the Linear function and takes the smaller value of the two. This is an alpha feature and should be used with caution. The following is an example of Super element sizing.

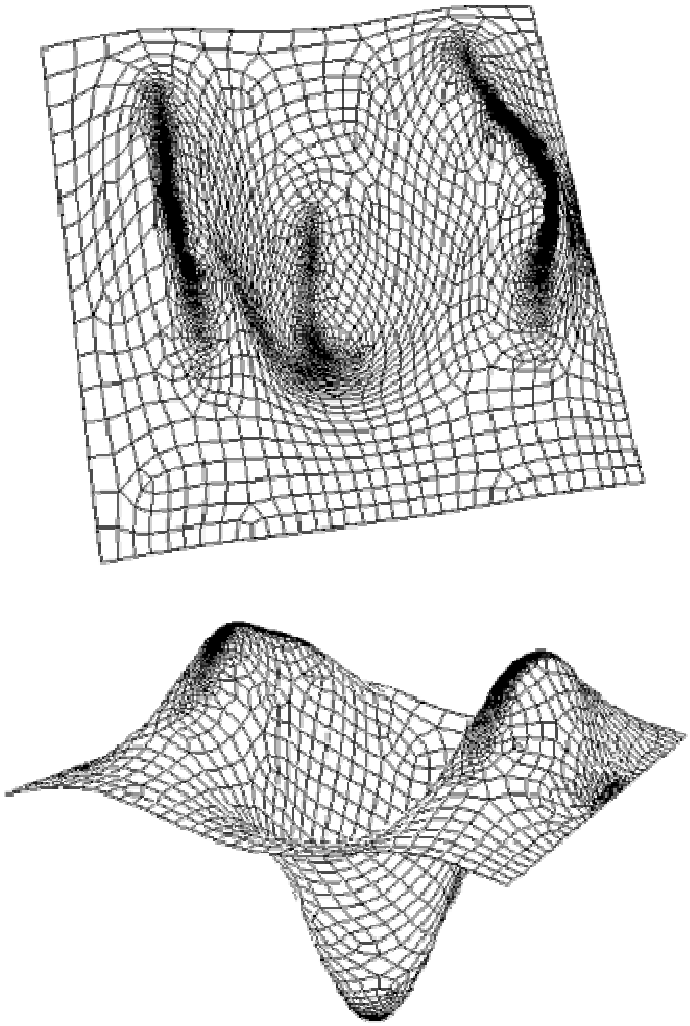


Figure 1. NURB mesh with super sizing function, 34 by 16 density

Test Sizing Function

The **Test** sizing function is a hardwired numerical function used to demonstrate the transitional effect of sizing function-based and adaptive paving. The function is a periodic function which is repeated in 50x50 unit intervals on a 2D surface in the first quadrant (x > 0, y > 0, z = 0). This is an alpha feature and should be used with caution. An example of a surface meshed with this sizing function is shown in Figure 1.

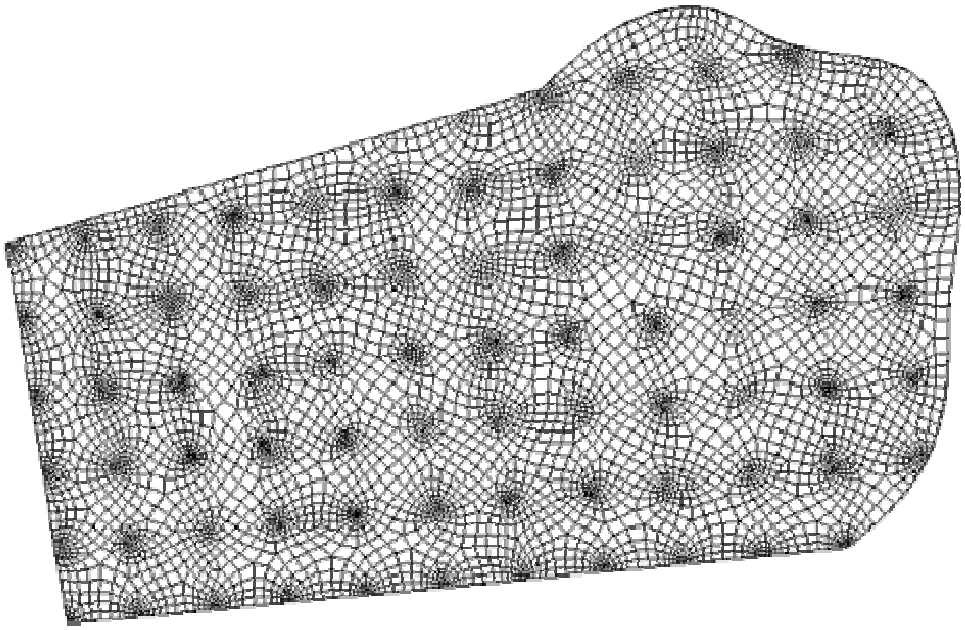


Figure 1. Test sizing function for spline geometry

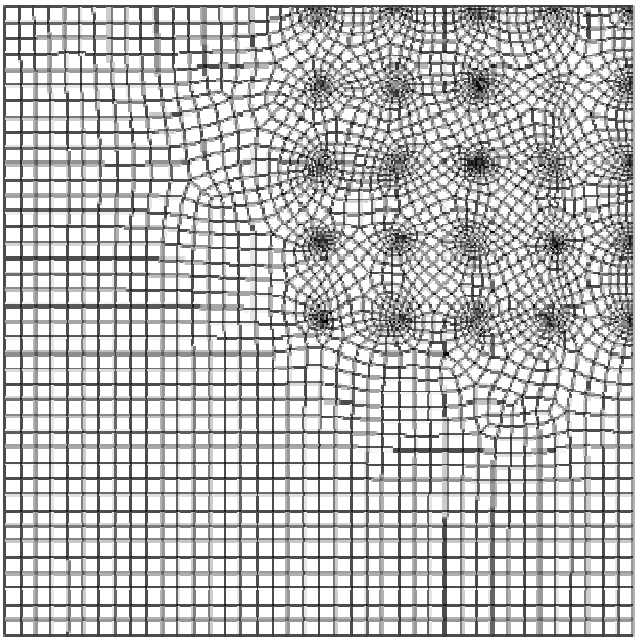


Figure 2. Test sizing function for square geometry

Transition

Applies to: Surfaces

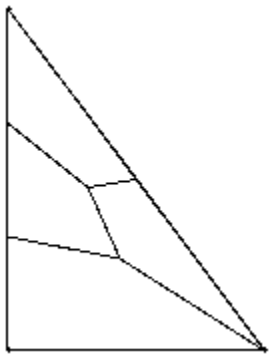
Summary: Produces a specified transition mesh for specific situations

Syntax:

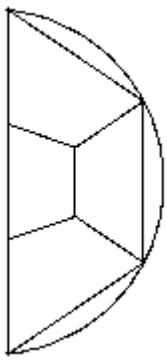
Surface <range> Scheme Transition
{Triangle|Half_circle|Three_to_one|Two_to_one|Convex_corner|Four_to_two} [Source Curve <id>]
[Source Vertex <id>]

Discussion:

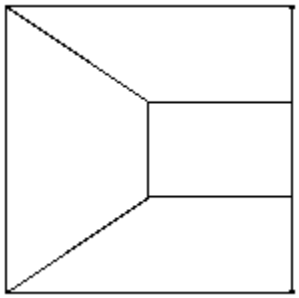
The *transition* scheme supplies a set of transition primitives which serve to transition a mesh from one density to another across a given surface. The six transition sub-types are demonstrated here.



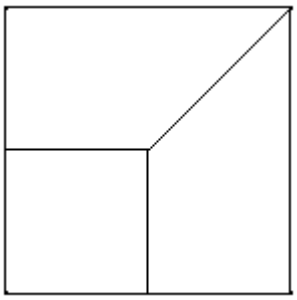
Scheme Transition **Triangl**e creates four quads in a triangle that has sides of three, two, and one intervals.



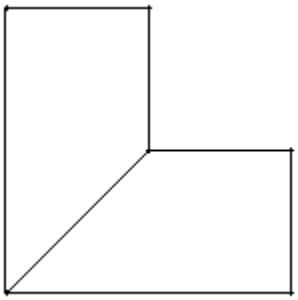
Scheme Transition **Half_circle** creates three intervals on the flat and three on the curved part of the half-circle, then creates four quads in the surface.



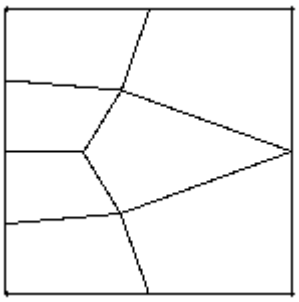
Scheme Transition **Three_to_one** creates four quads on a rectangular surface that has intervals of three, one, one, and one on its sides.



Scheme Transition **Two_to_one** creates three quads on a rectangular surface that has intervals of two, two, one and one on its sides :



Scheme Transition **Convex_corner** takes a six-sided block with a convex corner and connects that inner corner to the opposite one, creating two quads on the surface.



Scheme Transition **Four_to_two** creates seven quads on a rectangular surface that has intervals of four, two, two, and two on its sides.

The user also has the option of specifying a source curve and/or a source vertex. The rules for these specifications are as follows

- If both a curve and vertex are specified, the vertex must be on the curve.
- The Convex_corner sub-type does not allow a source curve.
- The Four_to_two sub-type does not allow a source vertex.
- The source curve will be the curve that will be given the fewest intervals.
- The source vertex will specify which corner will be used for the scheme, in cases where this makes sense (primarily in the Triangle, and Two_to_one cases).
- If none of the optional information is given, the program will assign the source curve to be the shortest one on the face, in keeping with the most probable

Untangle

Applies to: Triangular or Quadrilateral Surface Meshes Tetrahedral or Hexahedral Volume Meshes. Does not apply to Mixed Element Meshes such as those resulting from scheme HexTet.

Summary: Removes as many negative Jacobians from the mesh as possible by minimizing a certain objective function.

Syntax:

Surface <surface_id_range> Smooth Scheme Untangle [beta <double=0.02>] [cpu <double=10>]

Volume <volume_id_range> Smooth Scheme Untangle [beta <double=0.02>] [cpu <double=10>]

Related Commands:

Condition Number

Discussion:

The Untangle 'smoother' is designed to eliminate negative Jacobians from a given mesh by moving nodes to appropriate locations. If a mesh node is not involved in causing a negative Jacobian it will not be moved. If a mesh has no negative Jacobians, the Untangler will not move any of the nodes. This smoother is not magic: if an untangled mesh does not exist for the given mesh topology, the untangler will not untangle the mesh. Instead, it will do the best it can and exit gracefully. An untangled mesh produced by this smoother will often have poor shape quality; in that case it is recommended that untangling be followed by condition number smoothing. The untangle smoother is automatically called by the condition number smoother.

There is no "fixed/free" option with this command; boundary nodes are always held fixed. As a result, users should be aware that the volume untangler cannot succeed if the volume contains a surface mesh which contains a negative Jacobian. In that case, one must first remove the surface mesh negative Jacobians by invoking the surface Untangler and then invoke the volume Untangler.

The command above only sets the smoothing scheme; to actually smooth the mesh one must subsequently issue the command "smooth surface <surface_id_range>" or "smooth volume <volume_id_range>".

Stopping Criteria: Untangling will proceed until the objective function has been minimized or the optional user input "cpu" has been satisfied. The latter stopping criterion tells the code how many minutes to spend trying to untangle the mesh. The default value is 10 minutes. Optimization may also be halted by using "control-C" on your keyboard.

Beta Parameter: An optional user input parameter "beta" plays a role in determining the optimal mesh. Optimization proceeds until the minimum scaled Jacobian of the mesh is (roughly) greater than beta. To remove negative Jacobians one would need beta=0 (however, as a safety margin, we choose beta=0.02 as the default). To further improve the scaled Jacobian of the mesh, input a larger value of "beta". If a mesh with all scaled Jacobians greater than "beta" does not exist, optimization will continue until the cpu time stopping criterion has been met. Therefore, it is best not to use "beta" values too large (say, greater than 0.2) without also decreasing the cpu time limit.

To view a detailed report of the smoothing in progress issue the command "set debug 91 on" prior to smoothing the surfaces or volumes. You will get a synopsis of whether or not untangling is needed and whether the stopping criteria are satisfied. In addition the following printout information is given for each iteration of the conjugate gradient numerical optimization:

Iteration=n, Evals=m, Fcn=value1, dfmax=value2, time=value3 min_jsc=value4

n is the iteration count, **m** is the number of objective function evaluations performed per iteration, **value1** is the value of the objective function (this usually decreases monotonically), **value2** is the norm of the gradient (does not always decrease monotonically), and **value3** is the cumulative cpu time (in seconds) spent up to the current iteration. The minimum possible value of the objective function is zero; this value is attained only when the minimum scaled Jacobian of the mesh exceeds "beta". The **minimum scaled jacobian** is also reported.

Whisker Weave

Applies to: Volumes

Summary: Research algorithm for all-hexahedral meshing of arbitrary 3D volumes

Syntax:

Volume <range> Scheme Weave

Related Commands:

Pillow Volume <range>

{Volume|Surface|Curve} <range> Mesh [Fixed|Free]

Set AutoWeaveShrink [on|off]

Set Statelist [on|off]

Discussion:

Whisker Weaving (Tautges, 96; Tautges, 95; Folwell, 98) is a volume meshing algorithm currently being researched and is not released for general use. However, daring users may find the current form of the algorithm useful for mostly-convex geometries.

Whisker Weaving holds the promise of being able to fill arbitrary geometries with hexahedra that conform to a fixed surface mesh. The algorithm is based on the rich information contained in the Spatial Twist Continuum (STC) (Murdoch, 95), which is the grouping of the dual of an all-hexahedral mesh into an arrangement of surfaces called sheets. Given a bounding quadrilateral surface mesh, Whisker Weaving constructs sheets advancing from the boundary inward. The sheets are then modified so that the arrangement dualizes to a well defined hexahedral mesh. Once the primal hex-mesh is generated, interior node positions are generated by smoothing.

Examples of meshes generated using the whisker weaving algorithm are shown in the following figure.

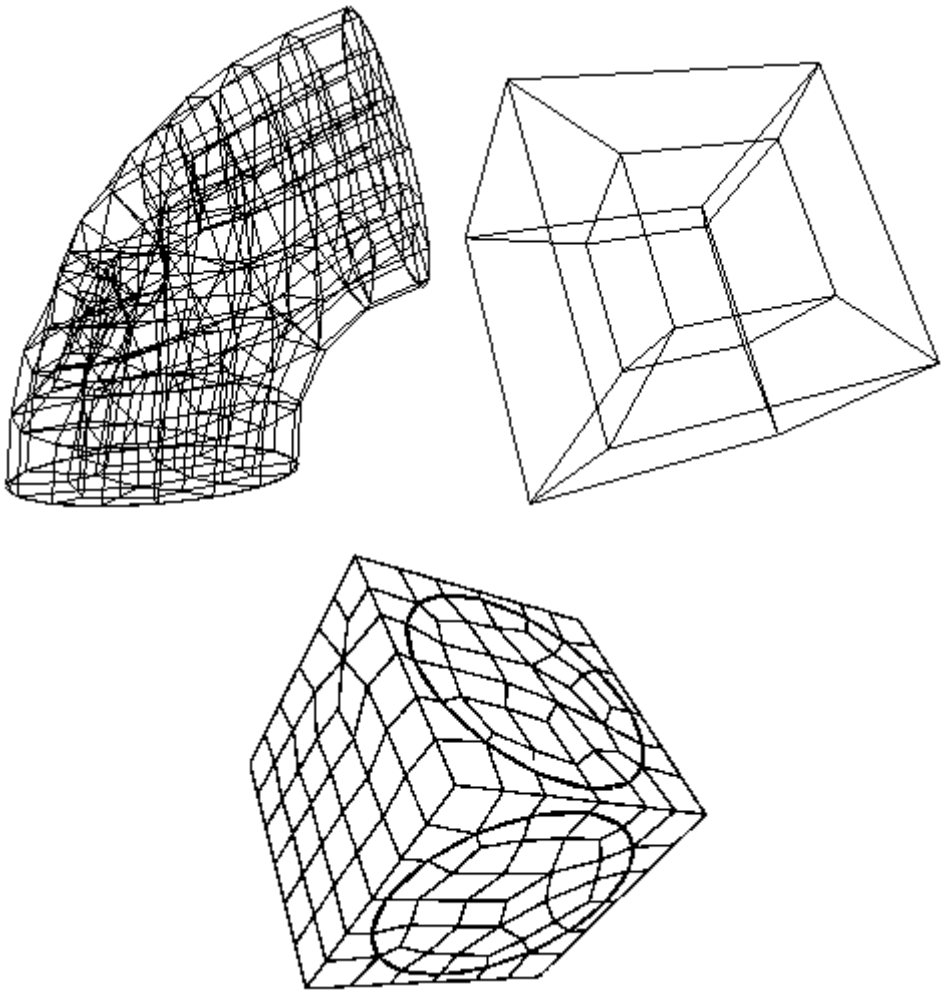


Figure 1. Some simple Whisker Weaving meshes with good quality

Whisker Weaving Basic Commands

The basic steps for meshing a volume with Whisker Weaving are the following:

- Set the meshing scheme for the volume to weave,

  Volume <range> Scheme Weave
- Mesh the volume, which generates hexes,

  Mesh Volume <range>
- Pillow the volume to remove certain additional degenerate hexes,

  Pillow Volume <range>
- and typically, smooth the mesh to improve quality, e.g.

  Volume <range> Smooth Scheme Condition Number

  Smooth Volume <range>

Whisker Weaving Options

Currently, Whisker Weaving relies on being able to perturb the bounding quadrilateral mesh. However, a bounding surface's mesh will not be changed if it is contained in another volume that is already meshed.

The user may also explicitly prevent Whisker Weaving from changing a bounding mesh by fixing it with the following command:

{Volume|Surface|Curve} <range> Mesh [Fixed|Free]

The user may select an optional control strategy that doesn't change the surface mesh by setting **AutoWeaveShrink** off, and setting **Statelist** on with the following commands:

Set AutoWeaveShrink [on|off]

Set Statelist [on|off]

Numerous developer commands are available for stepping through the algorithm, examining results, and toggling options. These are available via the command line help but are not detailed here.

Available Colors

All color commands in CUBIT require the specification of a color name. The following table lists the colors available in CUBIT at this time. The table lists the color number (#), color name, and the red, green, and blue components corresponding to each color, for reference.

Number	Color Name	Red	Green	Blue
0	black	0.000	0.000	0.000
1	grey	0.500	0.500	0.500
2	green	0.000	1.000	0.000
3	yellow	1.000	1.000	0.000
4	red	1.000	0.000	0.000
5	magenta	1.000	0.000	1.000

6	cyan	0.000	1.000	1.000
7	blue	0.000	0.000	1.000
8	white	1.000	1.000	1.000
9	orange	1.000	0.647	0.000
10	brown	0.647	0.165	0.165
11	gold	1.000	0.843	0.000
12	lightblue	0.678	0.847	0.902
13	lightgreen	0.000	0.800	0.000
14	salmon	0.980	0.502	0.447
15	coral	1.000	0.498	0.314
16	pink	1.000	0.753	0.796
17	purple	0.627	0.125	0.941
18	paleturquoise	0.686	0.933	0.933
19	lightsalmon	1.000	0.627	0.478
20	springgreen	0.000	1.000	0.498
21	slateblue	0.416	0.353	0.804
22	sienna	0.627	0.322	0.176
23	seagreen	0.180	0.545	0.341
24	deepskyblue	0.000	0.749	1.000
25	khaki	0.941	0.902	0.549
26	lightskyblue	0.529	0.808	0.980
27	turquoise	0.251	0.878	0.816
28	greenyellow	0.678	1.000	0.184
29	powderblue	0.690	0.878	0.902
30	mediumturquoise	0.282	0.820	0.800

31	skyblue	0.529	0.808	0.922
32	tomato	1.000	0.388	0.278
33	lightcyan	0.878	1.000	1.000
34	dodgerblue	0.118	0.565	1.000
35	aquamarine	0.498	1.000	0.831
36	lightgoldenrodyellow	0.980	0.980	0.824
37	darkgreen	0.000	0.392	0.000
38	lightcoral	0.941	0.502	0.502
39	mediumslateblue	0.482	0.408	0.933
40	lightseagreen	0.125	0.698	0.667
41	goldenrod	0.855	0.647	0.125
42	indianred	0.804	0.361	0.361
43	mediumspringgreen	0.000	0.980	0.604
44	darkturquoise	0.000	0.808	0.820
45	yellowgreen	0.604	0.804	0.196
46	chocolate	0.824	0.412	0.118
47	steelblue	0.275	0.510	0.706
48	burlywood	0.871	0.722	0.529
49	hotpink	1.000	0.412	0.706
50	saddlebrown	0.545	0.271	0.075
51	violet	0.933	0.510	0.933
52	tan	0.824	0.706	0.549
53	mediumseagreen	0.235	0.702	0.443
54	thistle	0.847	0.749	0.847
55	palegoldenrod	0.933	0.910	0.667

56	firebrick	0.698	0.133	0.133
57	palegreen	0.596	0.984	0.596
58	lightyellow	1.000	1.000	0.878
59	darksalmon	0.914	0.588	0.478
60	orangered	1.000	0.271	0.000
61	palevioletred	0.859	0.439	0.576
62	limegreen	0.196	0.804	0.196
63	mediumblue	0.000	0.000	0.804
64	blueviolet	0.541	0.169	0.886
65	deeppink	1.000	0.078	0.576
66	beige	0.961	0.961	0.863
67	royalblue	0.255	0.412	0.882
68	darkkhaki	0.741	0.718	0.420
69	lawngreen	0.486	0.988	0.000
70	lightgoldenrod	0.933	0.867	0.510
71	plum	0.867	0.627	0.867
72	sandybrown	0.957	0.643	0.376
73	lightslateblue	0.518	0.439	1.000
74	orchid	0.855	0.439	0.839
75	cadetblue	0.373	0.620	0.627
76	peru	0.804	0.522	0.247
77	olivedrab	0.420	0.557	0.137
78	mediumpurple	0.576	0.439	0.859
79	maroon	0.690	0.188	0.376
80	lightpink	1.000	0.714	0.757

81	darkslateblue	0.282	0.239	0.545
82	rosybrown	0.737	0.561	0.561
83	mediumvioletred	0.780	0.082	0.522
84	lightsteelblue	0.690	0.769	0.871
85	mediumaquamarine	0.400	0.804	0.667

Element Numbering

This appendix describes the element node and side numbering conventions used in Exodus II files written by CUBIT. This information is located here for convenience, but is identical to the information presented in the Exodus II manual; citation Schoof, 95

Node Numbering

The node numbering used for the basic elements is shown Figure 1. Specific element types of lower order just contain the number of nodes needed for those elements; for example, QUAD4 or QUAD elements use just the first four nodes shown for quadrilaterals in Figure 1.

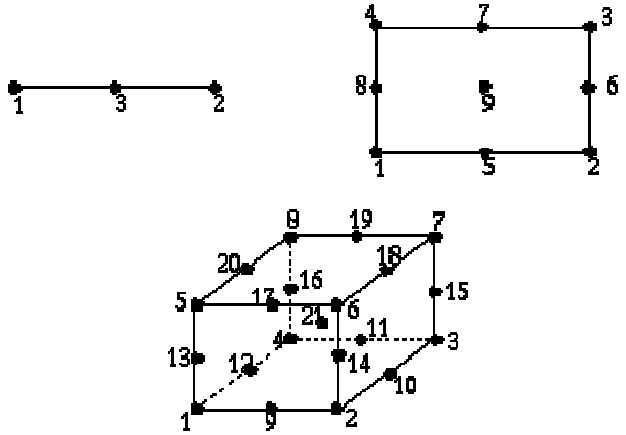


Figure 1. Local Node Numbering for CUBIT element types

Side Numbering

Element sides are used to specify boundary conditions that act over a length or area, for example pressure- or flux-type boundary conditions. Each element side is represented in the Exodus II format by an element number and the local side number for that element. The local side numbering for the basic elements is shown in Figure 2.

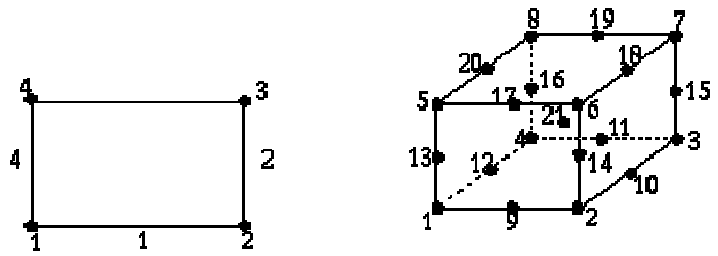


Figure 2. Local side numbering for CUBIT element types

Triangular Shell Element Numbering

A three-dimensional shell element with triangular topology will have the element type 'TRIANGLE'. This type can be modified for different element orders by appending the number of nodes onto the end of the type. For example, a 6-node shell could have the element type 'TRIANGLE6'. However, any element whose type begins with the 8 letters 'TRIANGLE' in upper, lower, or mixed case will refer to an element with a triangular topology. The element can exist in either three-space or two-space.

Attributes:
1. If the element exists in two-space, there are no required attributes.

2. If the element exists in three-space, there is one required attribute which is the thickness of the shell.

3. If the number of attributes is equal to the number of nodes in the connectivity of the element, then the attributes are assumed to specify the thickness of the element at each of the elements nodes. The ordering of the attributes matches the ordering of the elements nodes.

Node Ordering

The node ordering of the 3D triangle matches the node ordering of the 2D triangle as shown in Figure 3.

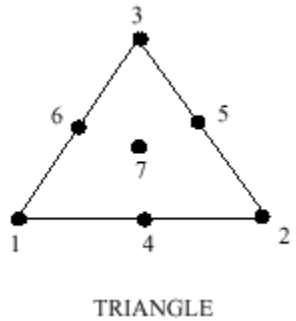


Figure 3. Local Node Numbering for CUBIT triangular element types

Side Set Side Ordering

The sideset side ordering is different for the element in the 2D and 3D instances.

In 2D, the sideset side ordering matches what is shown in Figure 4.

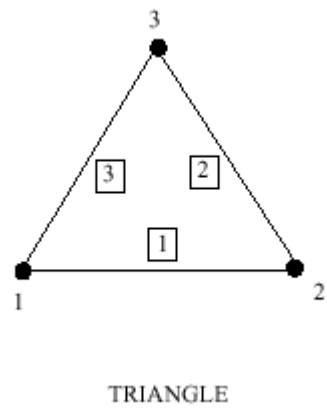


Figure 4. Local sideset numbering for CUBIT triangular element types

In 3D, the sideset side and node ordering is the same as for a quad shell except that there are only 3 or 6 nodes.

Then:

side 1 == {1,2,3}
side 2 == {3,2,1}
side 3 == {1,2}
side 4 == {2,3}
side 5 == {3,1}

If it is a higher order triangular shell (6 or 7 nodes), then the higher-order nodes are added on to the end of the above:

side 1 == {1,2,3,4,5,6,7}
side 2 == {3,2,1,6,5,4,7}
side 3 == {1,2,4}
side 4 == {2,3,5}
side 5 == {3,1,6}

Mesh Editing

Cubit provides a set of capabilities to take an existing mesh and edit it by modifying its mesh connectivity, nodal position, or genesis data. CAD solid model geometry is not necessary. This capability is useful for design studies, or adding detail to a model based on new understanding of the relevant features and physics. It also provides a graphical alternative to many GJoin and Grepos capabilities.

Model Acquisition

Import the mesh as mesh based geometry, or import the CAD solid model and then import the mesh and automatically associate it to the solid model.

In either case, all of the genesis entities are retained. After editing, the mesh can be re-exported, and the retained entities are exported as well. Blocks, as well as nodesets and sidesets can be edited by adding or removing individual geometric or mesh entities.

Manipulating Topology and Connectivity: Imprinting and Merging and Unmerging

You can import and export only part of a model, if desired, by specifying the blocks to be imported or exported. The mesh-based model can be unmerged on import, to facilitate editing just part of the mesh. (mesh import, and mesh export.)

In order to retain the geometric topology in the exported mesh, file, set the nodeset associativity on. This works when exporting mesh from either mesh based geometry or solid model geometry. To force a collection of mesh entities to become a geometric entity upon re-import, place them in a common nodeset or sideset. E.g.

```
brick wid 10
vol 1 int 4
mesh vol 1
export mesh 'plain_brick.g'
set nodeset associativity on
export mesh 'assoc_brick.g'
reset
# only one surface
import mesh geom 'plain_brick.g' feature_angle 0
reset
# six surfaces
import mesh geom 'assoc_brick.g' feature_angle 0
```

If the model in Cubit is merged, you can disconnect a piece of the mesh by unmerging the geometry. This piece can be manipulated separately, then re-merged with the rest of the model.

Meshed geometry can be merged, providing the mesh and geometry of the two objects have the same topology. If the node positions are different, they will be aligned during the merge process. E.g.

```
brick wid 10
brick wid 10
body 2 move 10 0 0
vol all interval 4
mesh volume all
surface 6 smooth scheme randomize
smooth surface 6
display
merge all
unmerge all
draw volume 2
```

If you have disconnected, meshed volumes, but the positions of the nodes and edges match, you can imprint the model using virtual geometry based on the mesh; see the new imprint mesh command. After doing so, the model may be merged normally. If you are going to remesh a volume, it can be important to imprint it before deleting its mesh. E.g.

```
brick wid 10
brick wid 10
body 2 move 10 5 0
vol all size 2
mesh vol all
imprint mesh volume all
merge all
```

If one or both of the volumes is unmeshed, it is still possible to imprint them manually using virtual geometry. (If both are ACIS-based unmeshed volumes, then use automatic solid model imprinting.) This manual virtual geometry imprinting can be tedious for large models, but will be automated in future Cubit versions. For a meshed volume, the basic idea is to partition it with its own mesh so that it's topology matches the other volume. For an unmeshed volume, the basic idea is to partition it with geometry. This geometry can be its own geometry, the geometry of another (meshed) volume, or new geometry created bottom-up from the positions of some mesh or geometry. Be sure to partition in a bottom up fashion, partitioning all of the curves before partitioning the surfaces. It is not yet possible for the user to partition volumes. The following is an example of the manual process.

```
# make the bodies, mesh one of them
brick wid 10
brick wid 10
body 2 move 10 5 0
vol 1 int 6
mesh vol 1

# partition the meshed surface
partition create surface 6 edge 396 398 400 402 404 405

# partition the unmeshed surface
# it is important to partition the curves first
partition create curve 15 vertex 2
partition create curve 19 vertex 5
partition create surface 10 curve 12
```

```
# merge
merge all
```

Options for mesh transfer

In order to get both objects meshed with compatible meshes, you can transfer the mesh from one object to another by copying the mesh, or exporting an interface mesh and importing it onto another geometry. It is possible to delete the mesh of a mesh-based volume, then import a new mesh onto it. When importing a mesh, part of the geometry may be already meshed, in which case the meshes will be merged.

Geometry (and Mesh) Manipulation

Mesh-based geometry, and geometry that is meshed, can be moved or otherwise transformed within Cubit. Mesh (including mesh-based geometry) can also be scaled on input or output by specifying a transformation.

You can create new pieces of geometry, either mesh-based or solid-model based geometry. Mesh-based and solid-model based geometry can be imprinted and merged with each other, as described above.

Mesh Manipulation

You can change the element order of blocks. The new element type will be used when the mesh is exported.

You can refine the mesh by dicing, refinement or by deleting the mesh, resizing, and remeshing.

You can reposition the mesh by smoothing.

FullHex vs. NodeHex Representation

CUBIT has two different internal representations of hexes: FullHexes and NodeHexes. The NodeHex is a lighter weight data structure, but occasionally nodeset and sideset shortcomings can be overcome by using FullHexes. The user can select which type of hexes get created when generating or importing a volume mesh with the following command:

```
Set FullHex [Use] [on|OFF]
```

Using the FullHex representation increases the memory used to store a mesh by a factor of approximately five.

APREPRO

- APREPRO Syntax
- APREPRO Rules
- APREPRO Operators
- APREPRO Predefined Variables
- APREPRO Functions
- Additional Functionality
- APREPRO Journaling

Within CUBIT there is support for a programming language called *APREPRO* (An Algebraic Preprocessor for Parameterizing Finite Element Analyses). Included here is a summary of the *APREPRO* functionality included within CUBIT. This is a summary of the full APREPRO user's manual (PDF).

APREPRO Syntax

Within CUBIT, *APREPRO* expressions must be written inside of curly braces {}. For example, the following is a valid CUBIT command:

```
Curve 1 Size {sqrt(2.0)}
```

- this will set the mesh size on curve 1 to 1.414214....(the square root of 2)

APREPRO expressions can also exist on separate lines as follows:

```
#{_numSeat=30}
```

- this will set the variable _numSeat to be equal to 30
- instead of a # you can use $ (i.e., ${_numSeat=30}

As in the example, separate line expressions must exist within commented lines. There is an exception though - looping expressions must exist on non-commented lines. See Additional Functionality .

APREPRO Rules

The rules that *APREPRO* uses when identifying functions, variables, numbers, operators, delimiters, and expressions are described below:

1. Functions

Function names are sequences of letters and digits and underscores (_) that begin with a letter. The function's arguments are enclosed in parentheses. For example, in the line **atan2(a,1.0), atan2** is the function name, and **a** and **1.0** are the arguments. See APREPRO Functions for a list of the available functions and their arguments.

2. Variables

A variable is a name that references a numeric or string value. A variable is defined by giving it a name and assigning it a value. For example, the expression **a = 1.0** defines the variable **a** with the numeric value **1.0**; the expression **b= "A string"** defines the variable **b** with the value **"A string"**. Variable names are sequences of letters, digits, and underscores (_) that begin with either a letter or an underscore. Variable names cannot match any function name and they are case-sensitive, that is, **abc_de** and **AbC_dE** are two distinct variable names. A few variables are predefined, these are listed in APREPRO Predefined Variables. Any variable that is not defined is equal to 0. A warning message is output to the terminal if an undefined variable is used, or if a previously defined variable is redefined .

3. Numbers

Numbers can be integers like **1234**, decimal numbers like **1.234**, or in scientific notation like **1.234E-26**. All numbers are stored internally as floating point numbers.

4. Strings

Strings are sequences of numbers, characters, and symbols that are delimited by either single quotes (**'this is a string'**) or double quotes (**"this is another string"**). Strings that are delimited by one type of quote can include the other type of quote. For example, {**'This is a valid "string"'**}. Strings delimited by single quotes can span multiple lines; strings delimited by double quotes must terminate on a single line or a parsing error message will be issued.

5. Operators

Operators are any of the symbols defined in APREPRO Operators. Examples are + (addition), - (subtraction), * (multiplication), / (division), = (assignment), and ^ (exponentiation).

6. Delimiters

The delimiters recognized by APREPRO are: the comma (,) which separates arguments in function lists, the left curly brace (**{**) which begins an expression, the right curly brace (**}**) which ends an expression, the left parenthesis (which begins a function argument list, the right parenthesis) which ends a function argument list, the single quote (') which delimits a multi-line string, and the double quote (") which delimits a single-line string.

7. Expressions

An expression consists of any combination of numeric and string constants, variables, operators, and functions. Four types of expressions are recognized in APREPRO: algebraic, string, relational, and conditional.

8. Algebraic Expressions

Almost any valid FORTRAN or C algebraic expression can be recognized and evaluated by APREPRO. An expression of the form **a=b+10/37.5** will evaluate the expression on the right-hand-side of the equals sign and assign the value to the variable a. An expression of the form **b+10/37.5** will simply evaluate the expression. Variables can also be set on the command line prior to playing any journal files using the 'var=val' syntax. Only a single expression is allowed within the **{ }** delimiters. For example, **{x = sqrt(y^2 + sin(z))}, {x=y=z}**, and **{x=y} {a=z}** are valid expressions, but **{x=y a=z}** is invalid because it contains two expressions within a single set of delimiters.

9. String Expressions

APREPRO has very limited string support. The only supported operations are assigning a variable equal to a string (**a = "This is a string"**) or a function that returns a string, and concatenating two strings into another string (**a = "Hello" // " " // "World"**).

10. Relational Expressions

Relational expressions are expressions that return the result of comparing two expressions. A relational expression is either true or false. Relational expressions can only be used on the left-hand side of a conditional expression. A relational expression is simply two expressions of any kind separated by a relational operator. See Relational Operators.

11. Conditional Expressions

APREPRO recognizes a conditional expression of the form::

```
relational_expression ? true_exp : false_exp
```

where **relational_expression** can be any valid relational expression, and **true_exp** and **false_exp** are two algebraic expressions. If the relational expression is true, then the result of **true_exp** is returned, otherwise the result of **false_exp** is returned. For example, if the following command were entered:

```
#{a = (sind(20.0) > cosd(20.0) ? 1 : -1)}
```

then, a would be assigned the value **-1** since the relational expression to the left of the question mark is false. Both **true_exp** and **false_exp** are always evaluated prior to valuating the relational expression. Therefore, you should not write an equation such as

```
#{sind(20.0*a)>cosd(20.0*a) ? a=sind(20.0) : a=cosd(20.0)}
```

since the value of a can change during the evaluation of the expression. Instead, this equation should be written as:

```
#{a = (sind(20.0*a)>cosd(20.0*a) ? sind(20.0) : cosd(20.0))}
```

APREPRO Operators

The operators recognized by APREPRO are listed below.

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Boolean Operators
- String Operators

In the following table, the letters **a** and **b** can represent variables, numbers, functions, or expressions unless otherwise noted. The tables below also list the precedence and associativity of the operators. Precedence defines the order in which operations should be performed. For example, in the expression:

3 * 4 + 6 / 2

the multiplications and divisions are performed first, followed by the addition because multiplication and division have higher precedence than addition. The precedence is listed from 1 to 14 with 1 being the lowest precedence and 14 being the highest.

Associativity defines which side of the expressions should be simplified first. For example the expression: **3 + 4 + 5** would be evaluated as **(3 + 4) + 5** for left associativity, the expression **a = b / c** would be evaluated as **a = (b / c)** for right associativity.

1. Arithmetic Operators

Arithmetic operators combine two or more algebraic expressions into a single algebraic expression. These have obvious meanings except for the pre- and post- increment and decrement operators. The pre-increment and pre-decrement operators first increment or decrement the value of the variable and then return the value. For example, if **a = 1**, then **b=++a** will set both **b** and **a** equal to **2**. The post-increment and post-decrement operators first return the value of the variable and then increment or decrement the variable. For example, if **a = 1**, then **b=a++** will set **b** equal to **1** and a equal to **2**. The modulus operator **%** calculates the integer remainder. That is both expressions are truncated an integer value and then the remainder calculated. See the **fmod** function in Mathematical Functions, for the calculation of the floating point remainder. The tilde character ~ is used as a synonym for multiplication to improve the aesthetics of the *APREPRO* unit conversion system (however, the unit conversions system is not supported in CUBIT). It is more natural for some users to type **12~metre** than **12*metre**

Table 1. Arithmetic Operators

Syntax	Description	Precedence	Associativity
a+b	Addition	9	left
a-b	Subtraction	9	left
a*b, a~b	Multiplication	10	left
a/b	Division	10	left
a^b, ab**	Exponentiation	12	right
a%b	Modulus (remainder)	10	left
++a, a++	Pre-, post-incerement	13	left
--a, a--	Pre-, post-decrement	13	left

2. Assignment Operators

Assignment operators combine a variable and an algebraic expression into a single algebraic expression, and also set the variable equal to the algebraic expression. Only variables can be specified on the left-hand-side of the equal sign.

Table 2. Assignment Operators

Syntax	Description	Precedence	Associativity
a=b	The value of 'a' is set equal to 'b'	1	right
a+=b	The value of 'a' is set equal to a + b	2	right
a-=b	The value of 'a' is set equal to a - b	2	right
a+=b	The value of 'a' is set equal to a * b	3	right
a/=b	The value of 'a' is set equal to a / b	3	right
a^=b	The value of 'a' is set equal to a^b	4	right
a=b**	The value of 'a' is set equal to a^b	4	right

3. Relational Operators

Relational operators combine two algebraic expressions into a single relational expression. Relational expressions and operators can only be used before the question mark (?) in a conditional expression.

Table 3. Relational Operators

Syntax	Description	Precedence	Associativity
a<b	true if 'a' is less than 'b'	8	left
a>b	true if 'a' is greater than 'b'	8	left
a<=b	true if 'a' is less than or equal to 'b'	8	left
a<=b	true if 'a' is greater than or equal to 'b'	8	left
a==b	true if 'a' is equal to 'b'	8	left
a!=b	true if 'a' is not equal to 'b'	8	left

4. Boolean Operators

Boolean operators combine one or more relational expressions into a single relational expression. If **la** and **lb** are two relational expressions, then:

Table 4. Boolean Operators

Syntax	Description	Precedence	Associativity

1a || 1b	true if either 'la' or 'lb' are true.	6	left
1a && 1b	true if both 'la' and 'lb' are true.	7	left
!1a	true if 'la' is false.	11	left

5. String Operators

The only supported string operator at this time is string concatenation, which is denoted by //. If **a = "Hello"** and **b = "World"**, then:

c = a // " " // b

sets **c** equal to **"Hello World"**. Concatenation has precedence 14 and left associativity. Also see String Functions

APREPRO Predefined Variables

A few commonly used variables are predefined in *APREPRO*. These are listed below. The default output format is specified as a C language format string, see your C language documentation for more information. The default format and comment variables are defined with a leading underscore in their name so they can be redefined without generating an error message.

Table 1. Predefined Variables

Name	Value	Description
PI	3.14159265358979323846	pi
PI_2	1.57079632679489661923	pi/2
SQRT2	1.41421356237309504880	$\sqrt{2}$
DEG	57.2957795130823208768	180 /pi degrees per radian
RAD	0.01745329251994329576	pi/180 radians per degree
E	2.71828182845904523536	base of natural logarithm
GAMMA	0.57721566490153286060	euler-mascheroni constant[1]
PHI	1.61803398874989484820	golden ratio ($\sqrt{5}$ + 1)/2
VERSION	Varies, string value	current version of CUBIT
_ FORMAT	"%.10g"	default output format
_C_	"#"	default comment character

1 The euler-mascheroni constant is defined as the limit of 1 + 1/2 + ... + 1/s - log(s) as s approaches infinity.

Note that the output format is used to output both integers and floating point numbers. Therefore, it should use the %g format descriptor which will use either the decimal (%d), exponential (%e), or float (%f) format, whichever is shorter, with insignificant zeros suppressed. The table below illustrates the effect of different format specifications on the output of the variable PI and the value 1.0 . See the documentation of your C compiler for

Table 2. Effect of Various Output Format Specifications more information. For most cases, the default value is sufficient.

Format	PI Output	1.0 Output
%.10g	3.141592654	1
%.10e	3.1415926536e+00	1.0000000000e+00
%.10f	3.1415926536	1.0000000000
%.10d	1413754136	0000000000

APREPRO Functions

Several mathematical, CUBIT and string functions are implemented in *APREPRO*.

- Mathematical Functions
- CUBIT Functions
- String Functions

To cause a function to be used, you enter the name of the function followed by a list of zero or more arguments in parentheses. For example

```
sqrt(min(a,b*3))
```

uses the two functions **sqrt()** and **min()**. The arguments **a** and **b*3** are passed to **min()**. The result is then passed as an argument to **sqrt()**. The functions in *APREPRO* are listed below along with the number of arguments and a short description of their effect.

1. Mathematical Functions

The following mathematical functions are available in APREPRO.

Table 1. Mathematical Functions

Syntax	Description
abs(x)	Calculates the absolute value of **x**. |**x**|
acos(x)	Calculates the inverse cosine of **x**, returns radians
acosd(x)	Calculates the inverse cosine of **x**, returns degrees
acosh(x)	Calculates the inverse hyperbolic cosine of **x**

asin(x)	Calculates the inverse sine of **x**, returns radians
asind(x)	Calculates the inverse sine of **x**, returns degrees
asinh(x)	Calculates the inverse hyperbolic sine of **x**
atan(x)	Calculates the inverse tangent of **x**, returns radians
atan2(y,x)	Calculates the inverse tangent of **y/x**, returns radians
atan2d(x)	Calculates the inverse tangent of **x**, returns degrees
atand(y,x)	Calculates the inverse tangent of **y/x**, returns degrees
atanh(x)	Calculates the inverse hyperbolic tangent of **x**
ceil(x)	Calculates the smallest integer not less than **x**
cos(x)	Calculate the cosine of **x**, with **x** in radians
cosd(x)	Calculate the cosine of **x**, with **x** in degrees
cosh(x)	Calculates the hyperbolic cosine of **x**
d2r(x)	Converts degrees to radians.
dim(x,y)	Calculates x - min(**x,y**).
dist(x_1,y_1, x_2,y_2)	Calculates distance from **x_1,y_1** to **x_2,y_2**
exp(x)	Calculates e^x (Exponential)
floor(x)	Calculates the largest integer not greater than x.
fmod(x,y)	Calculates the floating-point remainder of x/y.
hypot(x,y)	Calculates **sqrt**(x^2+y^2)
int(x), [x]	Calculates the integer part of **x** truncated toward 0.
julday(mm, dd, yy)	Calculates the Julian day corresponding to **mm**/**dd**/**yy**.
juldayhms (mm, dd, yy, hh, mm, ss)	Calculates the Julian day corresponding to **mm**/**dd**/**yy** at **hh**:**mm**:**ss**
lgamma(x)	Calculates log(G(***x***))
ln(x), log(x)	Calculates the natural (base e) logarithm of **x**.
logp1(x)	Calculates log(1+**x**)

log10(x)	Calculates the base 10 logarithm of **x**.
max(x,y)	Calculates the maximum of **x** and **y**.
min(x,y)	Calculates the minimum of **x** and **y**.
polarX(r,a)	Calculates ***r*** ´ **cos**(***a***), **a** is in degrees
polarY(r,a)	Calculates ***r*** ´ **sin**(***a***), **a** is in degrees
r2d(x)	Converts radians to degrees.
rand(xl,xh)	Calculates a random number between **xl** and **xh**.
sign(x,y)	Calculates **x** ´ sgn(***y***)
sin(x)	Calculates the sine of **x**, with **x** in radians.
sind(x)	Calculates the sine of **x**, with **x** in degrees.
sinh(x)	Calculates the hyperbolic sine of **x**
sqrt(x)	Calculates the square root of **x**.
tan(x)	Calculates the tangent of **x**, with **x** in radians.
tand(x)	Calculates the tangent of **x**, with **x** in radians.
tanh(x)	Calculates the hyperbolic tangent of **x**.
vangle(x1,y1, x2,y2)	Calculates the angle between the vector $x_1 i + y_1 j$ and $x_2 i + y_2 j$. returns radians.
vangled(x1,y1, x2,y2)	Calculates the angle between the vector $x_1 i + y_1 j$ and $x_2 i + y_2 j$. returns degrees.

2. CUBIT Functions

The following CUBIT Functions are available:

Table 2. CUBIT Functions

Syntax	Description
get_error_count()	Gets the current error count in CUBIT
set_error_count(val)	Sets the error count in CUBIT to given value
get_warning_count()	Gets the current warning count in CUBIT
set_warning_count	Sets the warning count in CUBIT to value

Id("name")	Find the ID of the entity with "name". Special case: use entity type ("body", "volume", "surface", "curve", "vertex", or "group") to get last created ID of that type
IntNum(id)	Finds the number of intervals on curve with given id
IntSize(id)	Finds the interval size on curve with given id
Length(id)	Finds the length of the curve with given id
Radius(id)	Finds the radius of the curve at its midpoint
Nx(id), Ny(id), Nz(id)	Finds the x, y or z coordinate of node with given id
Vx(id), Vy(id), Vz(id)	Finds the x, y or z coordinate of vertex with given id
NumInGrp("groupname")	Finds the number of entities in the given group

3.String Functions

A few useful string functions are available:

Table 3. String Functions

Syntax	Description
tolower(svar)	Translates all uppercase characters in **svar** to lowercase. It modifies **svar** and returns the resulting string.
toupper(svar)	Translates all lowercase character in **svar** to uppercase. It modifies **svar** and returns the resulting string.
tostring(x)	Returns a string representation of the numerical varaible **x**. The variable **x** is unchanged.
execute(svar)	**svar** is parsed and executed as if it were a line read from the input file. For example, **if svar = "b=sqrt(25.0)", then {execute(svar)}** returns the value **5** and sets **b = 5**. The expression **svar** is enclosed in delimiters prior to being executed and it must be a valid expression or an error message will be printed.
rescan(svar)	Similar to **execute(svar)**, except that **svar** is not enclosed in delimiters prior to being executed. For example, **if svar = "Create Vertex {1+5} {sqrt(5)} {sqrt(6)}", then {rescan(svar)}** would print: **Create Vertex 6 2.236067977 2.449489743.** The difference between **execute(sv1)** and **rescan(sv2)** is that **sv1** must be a valid expression, but **sv2** can contain zero or more expressions.

getenv(svar)	Returns a string containing the value of the environment variable **svar**. If the environment variable is not defined, an empty string is returned.
get_word(n,svar,del)	Returns a string containing the nth word of svar. The words are separated by one or more of the characters in the string variable del
word_count(svar,del)	Returns the number of words in **svar**. Words are separated by one or more of the characters in the string variable del
strtod(svar)	Returns a double-precision floating-point number equal to the value represented by the character string pointed to by **svar**.
error(svar)	Outputs the string **svar** to stderr and then terminates the code with an error exit status

The following example shows the use of some of the string functions.

#{t1 = "ATAN2"} {t2 = "(0, -1)"}

#{t3 = tolower(t1//t2)}

...The variable t3 is equal to the string atan2(0, -1)

#{execute(t3)}

...t3 = 3.141592654

The result is the same as executing {atan2(0, -1)}

This is admittedly a very contrived example; however, it does illustrate the workings of several of the functions. In the first example, an expression is constructed by concatenating two strings together and converting the resulting string to lowercase. This string is then executed.

The following example uses the rescan function to illustrate a basic macro capability in *APREPRO*. The example creates vertices in CUBIT equally spaced about the circumference of a 180 degree arc of radius 10. Note that the macro is 5 lines long (3 of the lines start with #, with the exception of the looping constructs - the actual journal file for this would not continue lines but would put each one on one long line).

#{num = 0} {rad = 10} {nintv = 10} {nloop = nintv + 1}

#{line = 'Create Vertex

{polarX(rad, (++num-1) * 180/nintv)}

{polarY(rad, (num-1)*180/nintv)}'}

{loop(nloop)}

#{rescan(line)}

{endloop}

Output:

Create Vertex 10 0

Create Vertex 9.510565163 3.090169944

Create Vertex 8.090169944 5.877852523

Create Vertex 5.877852523 8.090169944

Create Vertex 3.090169944 9.510565163

Create Vertex 6.123233765e-16 10

Create Vertex -3.090169944 9.510565163

Create Vertex -5.877852523 8.090169944

Create Vertex -8.090169944 5.877852523

Create Vertex -9.510565163 3.090169944

Create Vertex -10 1.224646753e-15

Note the loop construct to automatically repeat the rescan line. To modify this example to calculate the coordinates of 101 points rather than eleven, the only change necessary would be to set **{nintv=100}**.

APREPRO Additional Functionality

Additional APREPRO Functionality includes the following:

- File Inclusion
- Conditionals
- Loops

1. File Inclusion

APREPRO can read input from multiple files using the **include()** and **cinclude()** functions. If a line of the form:

{include(" filename")}

{include(string_variable)}

is read, *APREPRO* will open and begin reading from the file **filename**. A string variable can be used as the argument instead of a literal string value. When the end of the file is reached, it will be closed and *APREPRO* will continue reading from the previous file. The difference between **include** and **cinclude** is that if **filename** does not exist, **include** will terminate *APREPRO* with a fatal error, but **cinclude** will just write a warning message and continue with the current file. The **cinclude** function can be thought of as a conditional include, that is, include the file if it exists. Multiple include files are allowed and an included file can also include additional files. Approximately 16 levels of file inclusion can be used. This option can be used to set variables globally in several files. For example, if two or more input files share common points or dimensions, those dimensions can be set in one file that is included in the other files.

2. Conditionals

Portions of an input file can be conditionally processed through the use of the **{Ifdef(variable)}** or **Ifndef(variable)}** constructs. The syntax is:

#{Ifdef(variable)}

...Lines processed if **'variable'** is not equal to 0

#{Else}

...Lines processed if **'variable'** is equal to 0 or undefined

#{Endif}

#{Ifndef(variable)}

...Lines processed if **'variable'** is equal to 0 or undefined

#{Else}

...Lines processed if **'variable'** is not equal to 0

#{Endif}

The **{Else}** is optional. Note that if **variable** is undefined, its value is equal to zero. **Ifdef** constructs can be nested up to approximately 16 levels. A warning message will be printed if improper nesting is detected. **Ifdef(variable)}**, **{Ifndef(variable)}**, **{Else}**, and **{Endif}** are the only text parsed on a line. Text following these on the same line is ignored.

3. Loops

Repeated processing of a group of lines can be controlled with the **{loop(control)}**, **{endloop}** commands. The syntax is:

{loop(variable)}

...Process these lines **'variable'** times

{endloop}

Note that the loop constructs must not be put on comment lines (as other APREPRO single line commands - see APREPRO Syntax). Loops can be nested. A numerical variable or constant must be specified as the loop control specifier. You currently cannot use an algebraic expression such as **{loop(3+5)**}.

APREPRO Journaling

When using APREPRO, statements can be echoed to a journal file. Do do so, use the following command:

Set aprepro [ON|off].

Simply typing aprepro without an argument will display the current aprepro journaling setting.

For example,

bri x {2*5.0}

is journaled as

brick x {2*5.0}

if aprepro journaling is ON, or

brick x 10

if aprepro journaling is off. The default is **ON**.

APREPRO Comments

Comments are also journaled. This is useful for documenting aprepro definitions and descriptions.

Comments on the same line as a command get split into two separate lines in the journal file.

Significant Figures

When journal aprepro is **ON**, numbers are journaled exactly as they are entered. The maximum number of significant digits is determined by the command input.

When journal aprepro is **off**, numeric results of aprepro statements are journaled according to the maximum number of significant digits hard-coded into CUBIT, using the value of **DBL_DIG**.

Periodic Space Filling Models (Tile)

This appendix describes commands for producing good-quality meshes of models that tile space, such as polycrystalline materials models. Such models are often referred to as "periodic", but since that term already has a different meaning in Cubit, the keyword "tile" is used instead. Meshes may be smoothed across periodic boundaries. Periodic boundary conditions can be automatically set up, according to ALEGRA conventions (SAND99-2698).

Tile commands are alpha features and should be used with caution.

Initial setup

First import the model and merge the surfaces. Then mesh it with any method that will create meshes that match across the tile (periodic) boundary, say with scheme polyhedron or sweep. Once the mesh is created, specify the "tile vectors", which lets Cubit know that the nodes across the periodic boundaries are actually the same node:

```
Tile {x <period> | y <period> | z <period>}
[x <period>] [y <period>] [z <period>]
```

The 'period' you specify is actually the vector offset from one boundary to its match. Specify one tile command for each coordinate axis that the model is periodic in. E.g.

```
Tile x 1
Tile y 1
Tile z 1
```

You can see which nodes are matched to a given node by some combination of tile vectors with the following command:
Tile Debug Node <id>

If you later need to delete these tile vectors, use the following command:

```
Tile Off
```

Creating Nodesets

Once the tile vectors are specified, you can set up periodic boundary conditions that meet ALEGRA specifications. The command is:

```
Tile Nodeset <start_id>
```

This will create a nodeset for all combinations of tile vectors that actually connect nodes. The nodesets created will be reported to you. The nodesets will be consecutive starting with the given 'start_id', except that if there are no nodes for a particular combination there will be no nodeset and the id space will have a hole. To delete these nodesets, use the

```
Tile Off
```

command rather than the usual commands to delete nodesets.

Smoothing

Once a mesh has been created and the tile vectors have been specified, you can smooth the mesh and keep the periodic boundaries exactly offset by the tile vectors. Only hex meshes are currently supported. A variety of 3d smoothing schemes are supported, including laplac, equipotential, untangle, and condition number.

```
Smooth Volume <volume_id_range> [Global [Float <dim>] ]
```

Use "Global" if you are smoothing a collection of volumes. Use "float 3" if you want nodes on surfaces, curves, and vertices to be able to move off of their geometric owner. Use "float 2" if you want just nodes on curves and vertices to be able to move off of their owner (but stay on an owning surface). It is often useful to specify that some of the nodes are fixed using the "node position fixed" command.

Example

```
# make the geometry
#{brick_size=500}
brick wid {brick_size}
brick wid {brick_size}
body 2 move {brick_size} 0 0
brick wid {brick_size}
body 3 move {brick_size} {brick_size} 0
brick wid {brick_size}
body 4 move 0 {brick_size} 0
brick wid {brick_size}
body 5 move 0 0 {brick_size}
brick wid {brick_size}
body 6 move {brick_size} 0 {brick_size}
brick wid {brick_size}
body 7 move {brick_size} {brick_size} {brick_size}
brick wid {brick_size}
body 8 move 0 {brick_size} {brick_size}
merge all

# mesh it
vol all int 3
mesh vol all

# set the tiling vectors
tile x {brick_size*2}
tile y {brick_size*2}
tile z {brick_size*2}
tile debug node 256
tile debug node 245

# set the tiling nodesets
tile nodeset

# mess up the mesh quality
# volume all smooth scheme randomize
# smooth volume all
surface all smooth scheme randomize
smooth surface all
draw hex all

# fix the mesh quality
node in volume all position fixed
node in surface all position free
volume all smooth scheme laplac
# volume all smooth scheme untangle beta 0.08
smooth volume all global float 3
draw hex all
```

Troubleshooting Guide

If this happens...	Try This...
CUBIT gives me an error when attempting to import an IGES or STEP file	See Setting Up CUBIT for STEP or IGES tools.
The Windows version of CUBIT (Claro) crashes at startup or exhibits strange behavior	Try deleting the system registry entry for Claro:

	1. Start the Windows registry editor by going to Start->Run. Type in "REGEDIT" (without the quotes) in the Run dialogue and hit OK. **2.** Expand the tree HKEY_CURRENT_USER->Software. **3.** Click on Claro and hit the "Delete" key. **4.** Rerun Claro. **Warning** - this removes all customized settings in the GUI (docking, user icons, etc..).
CUBIT unexpectedly aborts while executing a command	While every effort has been made to make CUBIT bug-free, occasional bugs may still exist. To report a bug or suggest improvements to the program email cubit-dev@sandia.gov. A description of how to reproduce the problem along with any relevant journal or input files will assist the developers in tracking down the error. Corrected versions of CUBIT are available on a regular basis, so it may be worthwhile to download the latest version of CUBIT prior to reporting an error.
My Problem is not listed here	Check out the online CUBIT Users Junkyard for recent questions and answers.

References

Attaway, Stephen W.; Mello, Frank J.; Heinstein, Martin W.; Swegle, Jeffrey W.; Ratner, Julie A.; Zadoks, Rick Ian, "PRONTO3D users' instructions: a transient dynamic code for nonlinear structural analysis," Sandia Report SAND 98-1361 Sandia National Laboratories, Albuquerque, NM (1998)

Attaway S. W., unpublished, (1993)

Blacker, T. D., FASTQ Users Manual Version 1.2, SAND88-1326, Sandia National Laboratories, (1988)

Blacker, Ted D. "An Adaptive Finite Element Technique Using Element Equilibrium and Paving", American Society of Mechanical Engineers, Annual Meeting Dallas Texas, November 25-30, 1990, ASME, Nov 1990

Blacker, Ted D., "Paving: A New Approach To Automated Quadrilateral Mesh Generation", International Journal For Numerical Methods in Engineering, John Wiley, Num 32, pp.811-847, 1991

Blacker T.D. and Meyers R.J,."Seams and Wedges in Plastering:A 3D Hexahedral Mesh Generation Algorithm", Engineering with Computers, Springer Verlag, Vol 2, Num 9, pp.83-93, 1993

Brewer, M., L. Diachin, P. Knupp, T. Leurent, and D. Melander, "The Mesquite Mesh Quality Improvement Toolkit", Proceedings, 12th International Meshing Roundtable, 2003

Folwell, Nathan T. and Scott A. Mitchell, "Reliable Whisker Weaving via Curve Contraction", Proceedings, 7th International Meshing Roundtable, Sandia National Lab, pp.365-378, October 1998

Freitag, Lori A. and Patrick M. Knupp , "Tetrahedral Element Shape Optimization via the Jacobian Determinant and Condition Number", Proceedings, 8th International Meshing Roundtable, South Lake Tahoe, CA, U.S.A., pp.247-258, October 1999

Jones, R.E., QMESH: A Self-Organizing Mesh Generation Program, SLA - 73 - 1088, Sandia National Laboratories, (1974).

Knupp, Patrick M., "Next-Generation Sweep Tool: A Method For Generating All-Hex Meshes On Two-And-One-Half Dimensional Geomtries", Proceedings, 7th International Meshing Roundtable, Sandia National Lab, pp.505-513, October 1998

Knupp, Patrick M., "Winslow Smoothing On Two-Dimensional Unstructured Meshes", Proceedings, 7th International Meshing Roundtable, Sandia National Lab, pp.449-457, October 1998

Knupp, Patrick M., "Matrix Norms & The Condition Number: A General Framework to Improve Mesh Quality Via Node-Movement", Proceedings, 8th International Meshing Roundtable, South Lake Tahoe, CA, U.S.A., pp.13-22, October 1999

Knupp, P., "Achieving Finite Element Mesh Quality via Optimization of the Jacobian Matrix Norm and Associated Quantities, Part I", Int. J. Num. Meth. Engr.. 2000

Lovejoy, S. C. and R. G. Whirley, DYNA3D Example Problem Manual, UCRL-MA--105259, University Of California and Lawrence Livermore National Laboratory, (1990).

Melander, Darryl J., Timothy J. Tautges, Steven E. Benzley "Generation of Multi-Million Element Meshes for Solid Model-Based Geometries: The Dicer Algorithm" AMD-Vol. 220 Trends in Unstructured Mesh Generation, ASME, pp.131-135, July 1997

Meyers, Ray J., Timothy J. Tautges, Philip M. Tuchinsky, "The Hex-Tet Hex-Dominant Meshing Algorithm as Implemented in CUBIT", Proceedings, 7th International Meshing Roundtable, Sandia National Lab, pp.151-158, October 1998

Murdoch, Peter and Steven E. Benzley, "The Spatial Twist Continuum", Proceedings, 4th International Meshing Roundtable, Sandia National Laboratories, pp.243-251, October 1995

Oddy, A., J. Goldak, M. McDill, and M. Bibby "A Distortion Metric for Isoparametric Finite Elements" Transactions of the Canadian Soc. Mech. Engr., pp213-217, Vol 12, No 4, 1988.

Owen, Steven J. and David R. White, "Mesh-Based Geometry: A Systematic Approach to Constructing Geometry from the Nodes and Elements of a Finite Element Mesh", 10th International Meshing Roundtable, Sandia National Laboratories, pp. 83-96, October 2001

Parthasarathy V. N. et al, "A comparison of tetrahedron quality measures", Finite Elem. Anal. Des., Vol 15(1993), 255-261.

Robinson, J., "CRE method of element testing and Jacobian shape parameters, Eng. Comput., Vol. 4 (1987).

Ruppert, Jim , "A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation". Technical Report UCB/CSD 92/694, University of California at Berkely, Berkely California (1992)

Schoof, L. A.and Victor R. Yarberry, "EXODUS II A Finite Element Data Model", SAND92-2137, Sandia National Laboratories, (1995). Tautges, Timothy J. and Scott A. Mitchell, "Whisker Weaving: Invalid Connectivity Resolution and Primal Construction Algorithm", Proceedings, 4th International Meshing Roundtable, Sandia National Laboratories, pp.115-127, October 1995

Tautges, Timothy J., Ted Blacker, Scott A. Mitchell, "The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes", International Journal for Numerical Methods in Engineering, Wiley, Vol 39, pp.3327-3349, 1996

Taylor, L. M. and D. P. Flanagan, "Pronto 3D--A Three-Dimensional Transient Solid Dynamics Program", SAND87-1912, Sandia National Laboratories, (1989).

Tipton ,R. E., "Grid Optimization by Equipotential Relaxation", unpublished, Lawrence Livermore National Laboratory, (1990)

Tuchinsky, Philip M. and Brett W. Clark, "The HexTet Hex-Dominant Automesher: An Interim Progress Report" Proceedings, 6th International Meshing Roundtable, Sandia National Laboratories, pp.183-194, October 1997

Walton, D. J. and D. S. Meek, "A Triangular G1 Patch from Boundary Curves," Computer-Aided Design, Vol. 28 No. 2 pp. 113-123 (1996)

Watson, David F. , "Computing the Delaunay Tesselation with Application to Voronoi Polytopes", The Computer Journal, Vol 24(2) pp.167-172 (1981)

Wellman, Gerald W., "MAPVAR : a computer program to transfer solution data between finite element meshes", Sandia Report SAND 99-0466 Sandia National Laboratories, Albuquerque, NM (1999)

White, David R. and Paul Kinney, "Redesign of the Paving Algorithm: Robustness Enhancements through Element by Element Meshing", Proceedings, 6th International Meshing Roundtable, Sandia National Laboratories, pp.323-335, October 1997

White, David R. and Timothy J. Tautges, "Automatic Scheme Selection for Toolkit Hex Meshing", 2nd Symposium on Trends in Unstructured Mesh Generation, University of Colorado, Boulder, August 1999 (accepted for publication to special edition of International Journal for Numerical Methods in Engineering on Unstructured Mesh Generation, 2000)

Whiteley, M., D. White, S. Benzley and T. Blacker, "Two and Three-Quarter Dimensional Meshing Facilitators", Engineering with Computers, Springer-Verlag, Vol 12, pp.155-167, December 1996

Credits

Manager

- Ted Blacker, Manager, Computational Modeling Sciences Department (Org. 9226), Sandia National Laboratories

Project Board

- **Principal Investigator**: Steven J. Owen, Org. 9226,
- **SNL Support Manager**: Robert A. Kerr, Org. 9226,
- **SNL Product Manager**: Joe Bishop, Org. 9126, SNL

Research and Development

Computational Modeling Sciences Department, Org. 9226, Sandia National Laboratories, Albuquerque, NM

- Michael Borden
- Michael Brewer
- Byron W. Hanks
- Robert A. Kerr
- Darryl J. Melander
- Steven J. Owen
- Jason F. Shephard
- Matthew L. Staten
- Timothy J. Tautges

Elemental Technologies Inc., American Fork, UT

- Ray J. Meyers
- Corey Ernst
- Corey McBride
- Karl Merkley
- Randy Morris
- Mark Richardson
- Clinton Stimpson

Contractors

- Michael B. Stephenson, Provo, UT
- John Fowler, Albuquerque, NM

Caterpillar Co., Peoria, IL

- Ben Aga
- Joel Kopp
- Sam Showman
- Steven R. Storm
- Ramagy Yoeu

Brigham Young University, UT

- Steve Benzley (PI)

- Eve Crow
- Matt Earp
- Shane Ebbert
- Nathan Harris
- Mike Scott

Carnegie Mellon University, PA

- Kenji Shimada (PI)
- William Roshan Quadros Ved Vyas

Testing Staff

- Aaron Chomjak (Testing Lead), ETI, UT
- Weston Losinski, Org. 9226, Sandia National Laboratories

Documentation

- Sara Richards, Contractor, Champaign, IL

Administrative Assistant

- Lydia Koch , Org. 9226, Sandia National Laboratories

Quick Reference

Geometry | **File Import** | **Meshing** | **Genesis** | **Program** | **Entity Parsing** | **Groups** | **Graphics** | **Settings**

The following is brief overview of some of the most used command-line CUBIT commands.

GEOMETRY

Primitives

Brick X <> [Y <> Z <>]
Cylinder Radius <> Height <>
Frustum Z <> Radius <> [Top <>]
Frustum Z <> Maj Rad <> Min Rad <>
Prism Z <> Sides <> Rad <> [Maj <> Min <>]
Pyramid Height <> Sides <> Radius <>
Sphere Rad <> [Xpos] [Ypos] [Zpos] [Inn <>]
Torus Major Rad <> Minor Rad <>

Booleans

Unite <> [With <>] [keep]
Subtract <> From <> [keep]
Intersect <> [With <>] [keep]

Transformations

Body <> [Copy] Move <dx> <dy> <dz>
Move {} <> location {} <> [except [x] [y] [z]]
Rotate {} <> About {x| y| z|<> <> <>} Angle <>
Rotate {} <> About Vert <> Vert <> Angle <>
Rotate {} <> About Nor Of Surf <> Angle <> Body <> [Copy] Scale <> Body <> [Copy] Reflect {x| y| z|< x> <y> <z>}

Decomposition

Webcut {} <> Pla Vert <> [Vert]<> [Vert]<> ()
Webcut {} <> Plane Surf <> ()
Webcut {} <> Plane {xpla| ypla| zpla} [offs <>]
Webcut {} <> Tool [Body] <>
Webcut {} <> With Sheet {Body| Surf} <>
Webcut {} <> With Sheet Ext Fr Surf <>
Webcut {} <> Cyl Rad <> Axis {x| y| z| Vert <> Vert <>| <x><y><z>} [cent]
Options: [Noimprint| Imprint(default)], [Nomerge(default)| Merge], [group_ results] Section {} <> {{ xpla| ypla| zpla} [offs <>]} | Surf <>} [keep] [normal(default)| reverse]

FILE IMPORT

Import Acis 'filename'
Export Acis 'filename' [Body <>]
Import Mesh Geometry 'filename' (options)

MESHING

Mesh {} <>
Delete Mesh {} <> [Propagate]

Intervals

{} <> Interval {<> | Hard | Soft | Default}
{} <> Size {<> | Auto}
Match Intervals {} <> [Ass Grou [Onl| Infea]] [Seed Cur <>] [Map| Pave]

Mesh schemes

{} <> Scheme ...
Curve: bias, copy, curvature, dice, equal, stretch
Surface: auto, circle, copy, dice, hole, map, mirror, pave, pentagon, qtri, submap, triprimitive, trimap, trimesh, tripave
Volume: auto, copy, dice, map, sphere, submap, sweep, tetmesh, tetprimitive, thex
Smooth {} <>
{} <> Smooth Scheme ...

Smooth schemes

Curves: laplacian, randomize
Surface: centroid area pull, equipotential, laplacian, condition number, randomize, untangle, winslow
Volume: equipotential, laplacian, condition number, untangle, randomize

GENESIS

Block <> {Group| Vol| Surf| Curv} <> [Remove]
SideSet <> {Group| Curve} <> [Remove]
NodeSet <> {} <> [Remove]
Export Genesis 'filename'
Block <> Attribute <>
Block <> Element Type <type_>
Curves: bar[| 2| 3]| beam[| 2| 3]| truss[| 2| 3]
Surfaces: quad[| 4| 8| 9]| shell[| 4| 8| 9]| tri[| 3| 6| 7]
Volumes: hex[| 8| 20| 27]| pyr| tetra[| 4| 8| 10| 14] hexshell
SideSet <> Surf <> [Rem|[She][For| Rev| Both]]
SideSet <> Surf <> wrt Volume <>
Reset {Genesis | Nodesets | Sidesets | Blocks}

PROGRAM

Play 'filename'
Record {' filename' | stop}
Logging {off|on file <'filename'> [resume]}
Reset
Reset Genesis
Quit

ENTITY PARSING

Examples

Surface 1 2 3 4 to 6 by 2 ...
Curve all in Volume 2 ...
Draw Edge all in Hex 32
List Curve 1 to 50 except 2 4 6
Draw Sideset 1 2 3 Curve 3 to 5 Hex 2 4 6

GROUPS

Group <> {add| equals| remove| xor} {} <>
Group <> {inters| unite} grou <> with grou <>
Group <> subtract group <> from group <>

GRAPHICS

Default mouse buttons (command line)

B1 - rotate; B2 - zoom; B3 - pan
Control-B1: pick entity (In graph win: 0,1,2,3,4 - Pick vert, curv, surf, vol, body)

Shortcuts (focus in Graphics Window)

a Add to selection group
b Toggle Bounding Box on Click
c Clear "picked" Group
d Display 'picked' group, make it the selection
e Echo ID of selection to command line
f Assign function to mouse button
g List geometry of selection
h Print help
i Toggle visibility of selection
j/k Move slicing plane down/up
l List current selection (as if you typed 'list ...')
m/n List picked group/selection contents
p Toggle Persistent Wireframe
q Quit Current Mode (Exit slicing if slicing)
r Remove from 'picked' Group
s Toggle save-mesh on slice move
u Toggle mouse circle visibility
v Reset view
w Toggle Wireframe on click
x/y/z Slice along x/y/z-axis
Shift-Z Zoom on current selection
F1 Save view 1 Numbers: set what you're picking.
ESC Cancel current Action
Tab Next possible selection
Shift-Tab Previous possible selection

SETTINGS

Set Auto Sweep Scheme {Sw|Proj|Trans|Rot}
[set] Geometry Version <> (201, 300, 301, 401)
[set] Debug <index> [on|off]
[set] Debug <index> File <'filename'>
[set] Debug <index> Terminal
set Default Blocks [on|off|Volumes|Surfaces]
set Default Names [on|off] [set] Echo [on|off]
set Fix Duplicate Names [on|off]
set FullHex [Use] [on|OFF]
[set] Info [on|off]
[set] Journal [on|off]
set Keep Invalid Mesh [on/off]
[set] Logging {off|on file <'filename'> [resume]}
set Match Intervals Rounding {on|off}
set Match Intervals Fast {on|off}
set Node Constraint [ON|off]
[set] Paver Smooth Meth { Def | Smooth Sch}
[set] Paver Linearsizing {off|on}
set Replacement character '.|_|@'
[set] Scheme Auto Fuzzy [Tolerance] <degrees>
set {source|target} surface pattern '<pattern>'
set {Corner|End} Angle <degrees>
set Corner Weight <value>
set Turn Weight <value>
set Interval Weight <value>
set Large Angle Weight <value>
[set] Diagnostic {on|off}
set Suffix character '.|_|@'
[set] Smooth Meth {laplacian | isoparametric}
[set] Project Smooth [on|off]
[set] Warning [on|off]
[set] Smooth Iterations [default|<value>]
[set] Smooth Tol <> (Default = 0.05)